4

Final Technical Report

ONR Grant number: N00014-90-J-1353

(December, 1990 - June, 1992)

Project Title:

A Study of Neuronal Properties, Synaptic Plasticity and Network Interactions Using a Computer Reconstituted Neuronal Network Derived from Fundamental Biophysical Principles

P.I.: David C. Tam, Ph.D.

Division of Neuroscience
Baylor College of Medicine
1 Baylor Plaza
Houston, TX 77030

Present Address:
Center for Network Neuroscience
Department of Biological Sciences
University of North Texas
Denton, TX 76203
(817) 565-3261

E-mail address: dtam@sol.acs.unt.edu

Prepared for
Office of Naval Research
Computational Neuroscience Program
800 North Quincy Street
Arlington, VA 22217-5000

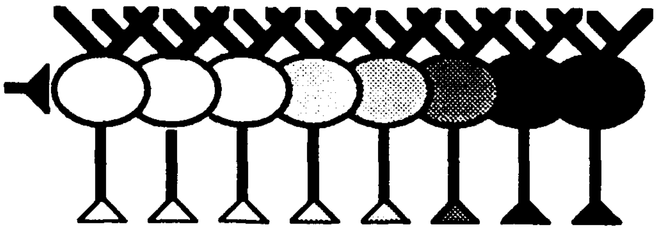


Table of Contents

Key Results and Transfer of Technology Resulted from the Current Grant..................2
Project Progress Report..................3
The Neural Simulator (MacNeuron)..................3
(1) Numerical Simulation Environment..................4
(2) Graphical Plotting Display of Simulation Results..................4
Invention of the Interspike Interval Decoding Artificial Neural Network..................5
New Multiple Spike Train Analysis Method..................5
Biological Neural Network Experimental Setup..................6
Publications produced during the grant period (1989 - 1992):..................8
Abstracts produced during the grant period (1989 - 1992):..................10
Patent Awarded (1992):..................12

Attached Documents

Curriculum vita

Awarded Patent Application Serial No. 07/630,463

MacNeuron Neural Simulator Tutorial

MacNeuron Neural Simulator User's Manual

Reprints of published papers (enclosed for Scientific Officier only)

Key Results and Transfer of Technology Resulted from the Current Grant

(1) The patented interspike interval decoding neural network technology has already been transfer to the technology transfer company called BCM Technology (Baylor College of Medicine Technology). BCM Technology has done an assessment study on the neural network technology, and is studying the possibility of licensing the technology to the industry.

(2) The novel cross-interval vector multiple-neuron spike train analysis technique developed under this grant is intended to transfer the technology to a multi-channel neural signal acquisition and neural signal processing hardware manufacturer with software interfaces. The company is a Dallas based electronics company called *Spectrum Scientific.* The company current designs and manufactures multi-channel amplifiers and neural signal data acquisition modules. The modules come in banks of 32-channels, which are capable of simultaneous acquiring and processing data 32-channels at a time. Full-configured systems are capable of processing multiples of 32-channels, up to 256 channels and beyond. The cross-interval vector multi-channel spike train analysis technique developed by the PI will be incorporated into the multi-channel data acquisition system so that data can be processed in real-time.

(3) The cross-interval vector multi-neuron spike train analysis technique and other multi-unit spike train analysis techniques will be packaged into a collection of neural signal processing algorithms to be implemented on high-speed computers. The technology is intended to be transferred to a company called CNS (Computational Neural Systems) for development and implementation.

(4) The MacNeuron neural simulation program developed under this grant will be made available and distributed to all interested neuroscientists once the program has gone through the beta-testing phase. The generic neural simulator will be of great use to the scientific community to simulate biological neurons using to model various neuron types by the specifications of the detailed neural elements.

Project Progress Report

The Neural Simulator (MacNeuron)

The neural simulator (called *MacNeuron*) is implemented on the Macintosh computer. The neural simulator is capable of simulating the electrical properties of a biological neuron which can be connected together with other neurons to form a neural network. Two versions are available: one runs on the Macintosh II series computer with the math coprocessor installed, the other runs without the math coprocessor.

The neural simulator allows users to build a compartmental neuron, simulate the generation of action potentials in neurons. It can record any variables associated with the neuron for graphical display and analysis. The variables include voltage, time and ionic concentration, etc. Each of these variables can be plotted on the graphical charts. The uniqueness of this neural simulator is the ability to "observe" any internal variables of the neural system during simulation, in contrast to other neural simulator where only a few predefined specific variables are observable.

A Tutorial Manual is written to guide new users on how to use the neural simulator. The tutorial also provides hands-on instructions of how to construct a neuron and a network using the simulator.

A User's Guide Manual is also written describing the neural simulator. It also serves as a reference manual describing its run-time environment. It also serves as a guide to use the neural simulator program.

The neural simulator is written in "an object-oriented programming language", which utilizes the latest software engineering technology for code development and code maintenance. This significantly reduces the development time and effort in the implementation of the neural simulator. It also facilitates the reusability of the code. Thus, it encourages other users to modify the existing code and/or add extended features to the neural simulator. This will significantly extend the life-time of the neural simulator. Furthermore, using an object-oriented design reduces the chances of bugs that may exist in the program, thus providing a quality assurance of the program results.

This neural simulator MacNeuron has implemented the generically changeable numerical algorithms for driving the neural simulator. Thus, it provides flexibility of using different efficient numerical algorithms without extensive recoding of the program. It is another unique advantage of this neural simulator over other existing neural simulator in its generalizabiltiy.

The program provides a window-based user-interface for building a network of neuron. The building process of the simulation environment is provided by the menu-driven window user-interface environment. A built-in text editor is provided within the simulation program for entering the script-file in describing the run-time environment. It also included the standalone plotting routines for displaying the simulation results.

Two specific contributions are made in the neural simulation program. First, the numerical simulation algorithms of neurons composed of patches of membrane (using the compartmental model) are implemented. The numerical integration routines are main computational engine for solving the systems of differential equations describing the biophysics of the ionic channels and receptors. Second, the graphical plotting of the simulation results is implemented. Any variables describing the simulation parameters can be plotted, thus providing visualization of the simulation results.

(1) Numerical Simulation Environment

The numerical engine for driving the neural simulator is implemented. The numerical integration algorithms used in this version is the conjugate gradient Euler methods. The numerical algorithms are required to solve the system of differential equations governing the dynamics of the ionic conductances, which is the key component in the neural simulator. Since our programming design is object-oriented, other numerical integration methods can be incorporated into our simulation program easily. In fact, the user will be able to choose the specific numerical methods at will when other numerical algorithms are implemented in the program. This is the essential feature in our simulator that most other simulators do not provide. Thus, it allows for flexibility for the user to choose whether to maximize the speed or accuracy of the simulation runs. Starting and stopping of simulation is done by a click of a "button".

(2) Graphical Plotting Display of Simulation Results

In the current implementation, the user can choose any two variables pair in the simulation and plot them graphically on a chart. Any variables used in the simulation can be "observed", and subsequently plotted on a graph. This provides flexibility to the user, so that the user can choose any variables, such as voltage, ionic concentration, gating activation variables, conductance, time, etc. can be observed during the simulation. To provide an intuitive approach to the user in selecting the plotting variable, the user needs only to "drag" any variable from a window which displays the list of parameter and "drop" that variable into the "observer" window and the "plot" window for plotting. Plotting of the results is done by clicking the "plot" button.

Thus, the current version of *MacNeuron* provides some basic building block for constructing a neuron for simulation. The incorporation of voltage-dependent conductance ionic gating channels on membranes for simulation and generation of action potentials can be accomplished currently in the simulator. Voltage step stimulation can also be accomplished for experimental manipulation.

Invention of the Interspike Interval Decoding Artificial Neural Network

A patent has been filed on the neural network signal decoding technology. The invention is *"An Interspike Interval Decoding Neural Network."* used to decode the time interval between firing of neurons from a serial representation to a parallel representation. The neural network uses a unique architectural hierarchical connectivity structure to process spike train signals. It also uses specific excitatory and inhibitory connections for processing of signals using the hardwired configuration. This unique design of cascaded connectivity allows the network to extract the interspike intervals of the input spike train and separate out the intervals into a topographical map. As a result, the serial input spike train signal is converted into a parallel distributed output spike train signal, where the topographical location of the output represents the specific decoded interspike interval of the original spike train.

A neural network using pulse-coded signal for processing is developed to decode interspike interval between the firing of action potentials in neurons. The neural network uses time-delays and appropriate excitatory and inhibitory connections for signal processing. With appropriate connections using a cascaded time-delay (or time-shifting) scheme, the signals are able to propagate to different neurons appropriately by the exact time interval for extracting the firing intervals. This neural network represents the outputs in a two-dimensional topographical map configuration, where the location of neurons which fire at the output layer represents the firing interval being decoded. The network, thus, able to extract serial-code into parallel-code. That is, it converts from serial representation of firing interval to parallel representation of firing by the location of the activated neurons.

The Multiple Spike Train Analysis Method

A new statistical multiple spike train analysis method, called cross-interval vector, has also been developed. This statistical method is used to detect the spatio-temporal correlation among many neurons. Since most conventional spike train analysis methods are designed to extract correlation between two to three neurons, this new method will provide quantitative analysis of a large number of neurons recorded from a biological neural network. The ability to analyze the signals generated by a large number of neurons is important in the understanding of how a biological neural network may operate, from which the principles of operation may be extracted.

This novel technique uses a vectorial measure to detect correlation between firing patterns of multiple neurons simultaneously. The statistic utilizes two random variables, the pre-cross interval and the post-cross interval, for the construction of the cross-interval vector. A cross-interval is the time interval between the occurrence of adjacent spikes in two different neurons. The statistics provided by the occurrence times of the spikes in these neurons will provide us with the estimate of the probability function of firing in these neurons. With the probability of firing in neurons relative to one another known, the stochastic firing characteristics of neural signals can be understood. The utilities of stochastic signal processing by neurons can be extracted for better designs of artificial neural network with extensive fault tolerant capability in the fuzziness of the signals used by the network.

Since this technique uses a vector to represent the preceding and succeeding firing intervals (cross-intervals), this cross-interval vector represents the timing relationships between the firings of any neurons. To compute the statistical average of the population, the resultant vector, which is the vectorial sum of the population, can be used to represent the present the timing relationship of all neurons relative to the reference neuron. Thus, this vectorial measure is used to detect any correlational relationship between any reference neuron and the rest of the population.

The vectorial statistical technique enables detection of correlated firing pattern between any number of neurons. As with conventional correlation techniques, the temporal correlation between the firing times of neurons can be deduced. In addition, with this new technique, not only the temporal correlation, but also the spatial correlation can be established. The spatial correlation among any number of neurons is an important attribute that is unique to neural network signal processing. With the combined spatio-temporal correlation, highly complex neural signal processing scheme can be extracted from a network of neurons.

Biological Neural Network Experimental Setup

A new biological neural network experimental setup is being set up currently to record from a large number (in excess of 64) of neurons simultaneously here at the Center for Network Neuroscience of the University of North Texas (after moving from Baylor College of Medicine). The experimental results will be incorporated with the simulation results to test the hypotheses of neural network processing.

The experimental setup is capable of recording from up to 64 microelectrodes photoetched on a silicon substrate. The closely spaced microelectrodes enable recording of many neurons simultaneously from a network of neurons. These neurons can be cultured and grown on the microelectrode plates for subsequent electrical recording and stimulation. Well-isolated electrical signals can be recorded from these electrodes. Networks of biological neurons can be cultured on these multi-microelectrodes, so that the electrical activity of these neurons can be monitored simultaneously. Furthermore, electrical stimulation of specific neurons in the network can be accomplished by delivering currents to the microelectrodes.

Successful electrical stimulation will provide a powerful experimental tool to test the hypothesis of biological network learning since it will provide a means for "training" the network with specific sequence, and observe the output of the network by recording the electrical activity of neurons from the microelectrodes.

Two types of electrodes will be used. A 64-channel microelectrode photoetched on transparent glass plate will be used to record from neuronal cell culture. A neural network can be grown on such electrode plate, which can then be recorded and stimulated simultaneously. Since the electrode is also transparent, optical imaging recording can be done simultaneously with the electrical recording. Such recording can be done *in vitro* on cell culture or brain slice.

Another setup will also be used for recording *in vivo*. A multi-stranded electrode bundle will be used to record from the cortices of animal implanted with this electrode. Thus psychophysical experiments on learning and memory can be done while recording from the activity of the neurons in the cortical network.

Publications produced during the grant period (1989 - 1992):

Tam, D. C. (1992) Vectorial phase-space analysis for detecting dynamical interactions in firing patterns of biological neural networks. Proceedings of the International Joint Conference on Neural Networks, June 1992. Vol.3 pp.97-102.

Tam, D. C. (1992) Novel cross-interval maps for identifying attractors from multi-unit neural firing patterns. In: Nonlinear Dynamical Analysis of the EEG.. (B. Jensen, ed.) World Scientific Publishing Co (in press)

Tam, D. C. (1992) A multi-neuronal vectoral phase-space analysis for detecting dynamical interactions in firing patterns of biological neural networks. In: Computational Neural Systems. (J. M. Bower, ed.) (in press)

Tam, D. C. and Hutson, R. K. (1992) An object-oriented paradigm for the design of realistic neural simulators. In: Computational Neural Systems. (J. M. Bower, ed.) (in press)

Tam, D. C. (1992) A novel vectorial phase-space analysis of spatio-temporal firing patterns in biological neural networks. Proceedings of the Simulation Technology Conference. (in press)

Tam, D. C. (1992) A generalizable object-oriented neural simulator for reconstructing functional properties of biological neuronal networks. Proceedings of the Simulation Technology Conference. (in press)

Tam, D. C. (1992) Objecı oriented programming techniques for implementing generalizable models. Proceedings of the Simulation Technology Conference. (in press)

Tam, D. C. (1992) A hybrid time-shifted neural network for analyzing biological neuronal spike trains. Progress in Neural Networks (O. Omidvar, ed.) Vol. 2, Ablex Publishing Corporation: Norwood, New Jersey. (in press)

Tam, D. C. (1991) Signal processing in multi-threshold neurons. In: Single Neuron Computation (T. McKenna, J. Davis, and S. F. Zornetzer, eds.) Academic Press, San Diego. pp. 481-501.

Tam, D. C. (1991) Signal processing by multiplexing and demultiplexing in neurons. In: Advances in Neural Information Processing Systems. (D. S. Touretzky, ed.), Morgan Kaufmann Publishers, San Mateo, California. pp. 282-288.

Alkon, D. L., Vogl, T. P, Blackwell, K. T. and Tam, D. C. (1991) Memory function in neural and artificial networks. In: Neural Network Models of Conditioning and Action. (M. L. Commons, S. Grossberg, J. E. R. Staddon, eds.) pp. 1-11. Lawrence Erlbaum Associates: Hillsdale, New Jersey.

Tam, D. C. (1990) Decoding of firing intervals in a temporal-coded spike train using a topographically mapped neural network. Proceedings of the International Joint Conference on Neural Networks, June, 1990. Vol. 3, pp. III-627–632.

Tam, D. C. (1990) Temporal-spatial coding transformation: Conversion of frequency-code to place-code via a time-delayed neural network. Proceedings of the International Joint Conference on Neural Networks (H. Caudill, eds.), Jan., 1990. Vol. 1, pp. I-130–133.

Tam, D. C. (1989) The physiological basis and implications of differential motor activation. Behavioral and Brain Sciences Vol. 12, pp. 669.

Tam, D. C. (1989) A positive/negative reinforcement learning model for associative search network. Proceedings of the First Annual IEEE Symposium on Parallel and Distributed Processing, 1989, (B. Shirazi, ed.) pp. 300-307.

Tam, D. C. and Perkel, D. H. (1989) Quantitative modeling of synaptic plasticity. In: The Psychology of Learning and Motivation: Computational Models of Learning in Simple Neural Systems, (R. D. Hawkins and G. H. Bower, eds.) Vol. 23, pp. 1 - 30. Academic Press: San Diego.

Tam, D. C. and Perkel, D. H. (1989) A model for temporal correlation of biological neuronal spike trains. Proceedings of the IEEE International Joint Conference on Neural Networks 1989. Vol. 1, pp. I-781–786.

Abstracts produced during the grant period (1989 - 1992):

Tam, D. C. (1992) A new multi-conditional statistical measure for detecting spatiotemporally correlated firing patterns in multiple spike trains. Society for Neuroscience Abstract. (in press)

Kenyon, G. T. and Tam, D. C. (1992) Using the assymptotic kolmogorov entropy to identify deterministic structures in multi-unit spike trains. Society for Neuroscience Abstract. (in press)

Zouridakis, G. and Tam, D. C. (1992) Multi-unit spike discrimination using wavelet transforms. Society for Neuroscience Abstract. (in press)

Tam, D. C. and Kenyon, G. T. (1992) Novel Cross-Interval Maps for Identifying Attractors from Multi-Unit Neural Firing Patterns. Second Annual Conference on Nonlinear Dynamical Analysis of the EEG. (in press).

Tam, D. C. and Kenyon, G. T. (1992) A vectorial statistical method for detecting correlated firing patterns in neurons. Biophysical Society Abstract. Vol. 16, p.A175.

Kenyon, G. T. and Tam, D. C.(1992) An object-oriented paradigm for simulating physiclogical processes in extended biological structures. Biophysical Society Abstract. Vol. 16, p.A175.

Tam, D. C. (1992) Methods for investigating the neurophysiological functions of the central nervous system. Society of Chinese Bioscientists Symposium Abstract.

Tam, D. C. (1992) A vectorial statistical measure for detecting temporally correlated firing patterns in multiple spike trains. Society of Chinese Bioscientists Symposium Abstract.

Tam, D. C. and Kenyon, G. T. (1991) A novel vectorial measure for detecting temporally correlated firing patterns in multiple spike trains. Society for Neuroscience Abstract. Vol. 17, p. 125.

Boney, D. G., Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating inter-connected neural systems. Society for Neuroscience Abstract. Vol. 17, p. 126.

Tam, D. C. (1991) A vectorial statistical method for analyzing stochastic firing patterns in large numbers of neurons in parallel. The Second Keck Symposium on Computational Biology Abstract.

Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating neurophysiological processes. The Second Keck Symposium on Computational Biology Abstract.

Tam, D. C. (1990) Functional significance of bi-threshold firing of neurons. Society for Neuroscience Abstract. Vol. 16, p. 1091.

Tam, D. C. (1990) Hebbian synapse and its relation to cross-correlation function in associative conditioning learning. Eighth Annual Conference on Biomedical Engineering Research in Houston. p. 5.

Tam, D. C. (1990) Visual-motor integration: What role does cerebellar cortical neurons participate in movement control in monkeys? Eighth Annual Conference on Biomedical Engineering Research in Houston. p. 39.

Tam, D. C. and McMullen, T. A. (1989) Hebbian synapses as cross-correlation functions in delay line circuitry Society for Neuroscience Abstract Vol. 15, p. 777.

Alkon, D. L., Vogl, T. P., Blackwell, K. T. and Tam, D. C. (1989) Pattern recognition and storage by an artificial network derived from biological systems. Neural Network Models of Conditioning and Action: the Twelfth Symposium on Models of Behavior at Harvard University.

Patent awarded (1992):

An Interspike Interval Decoding Neural Network.

Patent Serial No. 07/630,463. (waiting for final patent number to be assigned by Patent office)

Patent description:

A multi-layered artificial neural network designed to decode the interspike intervals of a spike train (a time series of pulse-coded signals) by mapping the one-dimensional time-series signal (serial signal) into a topographically mapped spatio-temporal (two-dimensional) output signal (parallel signal) using a specific hardwired cascaded architectural design with excitatory and inhibitory connections.

David C. Tam

Center for Network Neuroscience
Department of Bioligical Sciences
University of North Texas
Denton, TX 76203

(817) 565-3261 (*office*)
(817) 565-4136 (*FAX*)

E-mail: dtam@sol.acs.unt.edu

HOME ADDRESS	809 Ector St. Denton, TX 76201	**Mailing address:**	P. O. Box 6666 Denton, TX 76203
HOME PHONE	(817) 380-0377		

EDUCATION

Degrees Earned	Institution	Year
Ph. D. in Physiology (minor in Physics)	University of Minnesota, Minneapolis, MN	1987
B. Sc. in Physics	University of Minnesota, Minneapolis, MN	1984
B. Sc. in Astrophysics	University of Minnesota, Minneapolis, MN	1984
B. Sc. in Computer Science	University of Minnesota, Minneapolis, MN	1980

PROFESSIONAL WORK EXPERIENCE

Job Title	Address	Year
Asstistant Professor	Center for Network Neuroscience Department of Biological Sciences University of North Texas Denton, TX 76203	1992 - present
Res. Asst. Professor	Division of Neuroscience Baylor College of Medicine Houston, TX 77030	1989 - 1992
Visiting Scientist	Neural Systems Section, NINDS National Institute of Health Bethesda, MD 20892	1989
Post-doctoral Researcher	Dept. of Physiology & Biophysics University of California Irvine,CA 92717	1988 - 89
Post-doctoral Researcher	Dept. of Psychobiology University of California Irvine,CA 92717	1987 - 88
Application Programmer	Neurosurgery Dept., University of Minnesota Minneapolis, MN 55455	1978-84,85-87
System Programmer	Cell Biology & Neuroanatomy Dept. University of Minnesota Minneapolis, MN 55455	1985
Research Assistant	Institute of Child Development University of Minnesota Minneapolis, MN 55455	1984 - 85
Teaching Assistant	Physiology Dept. University of Minnesota Minneapolis, MN 55455	1981 - 87
Research Assistant	Metallurgy Dept. University of Minnesota Minneapolis, MN 55455	1978
Computer Consultant	Applied Statistics Dept. University of Minnesota St. Paul, MN 55455	1977 - 78

GRANT AWARDS

Office of Naval Research — *Grant number N00014-90-J-1353*

"A Study of Neuronal Properties, Synaptic Plasticity and Network Interactions Using a Computer Reconstituted Neuronal Network Derived from Fundamental Biophysical Principles"

12/89 - 12/92 — $259,985

USPHS BRSG — *Grant number RR05425-30*

"Synaptic Interactions in Neuronal Networks"

6/91 - 3/92 — $10,000

PENDING GRANT

Office of Naval Research — *Grant number N00014-90-J-1353*

"Dynamics of Neural Processing in Inter-Connected Neurons"

11/92 - 11/94 — $200,000

PATENT AWARDED

An Interspike Interval Decoding Neural Network. — Serial No. 07/630,463.

ASSOCIATE EDITOR

Journal of Artificial Neural Networks	1991 - present
Progress in Neural Networkss	Volume Editor

JOURNAL REVIEWER

Journal of Theoretical Biology
Brain Research
Behavioral and Brain Sciences
Progress in Neural Networks
Cerebral Cortex

PROFESSIONAL SOCIETY MEMBERSHIP

American Association for the Advancement of Science	(AAAS)	1992 - present
Biophysical Society		1989 - present
Apple Programmers and Developers Association	(APDA)	1989 - present
Mathematical Association of America	(MAA)	1988 - present
Behavioral and Brain Sciences Associate	(BBS)	1988 - present
International Neural Network Society	(INNS)	1987 - present
Institute of Electrical and Electronics Engineers	(IEEE)	1985 - present
Society for Neuroscience		1983 - present
Association of Computing Machinery	(ACM)	1979 - present

TEACHING EXPERIENCE:

Computational Neuroscience Special Topics Course	at Baylor College of Medicine	1991
Computational Neuroscience Journal Club	at Baylor College of Medicine	1990 - 1992

Teaching Assistant in:

Neurophysiology recitation for medical students	at University of Minnesota	1980 - 1987
Neurophysiology lab for graduate students	at University of Minnesota	1980 - 1987
Physiology lab for dental students	at University of Minnesota	1980 - 1987
Physiology lab for pharmacy students	at University of Minnesota	1980 - 1987
Physiology lab for dental hygiene students	at University of Minnesota	1980 - 1987

RESEARCH INTERESTS AND EXPERTISE

The primary research goal is to understand the parallel signal processing capabilities of biological neurons, neural networks and their functions in the central nervous system. The research is focused on computational neuroscience and the analysis of the signals encoded by multiple neurons in a network. Two approaches are employed to study the function of the neural processing system: theoretical modeling and experimental analysis. Theoretical studies include developing computational models of the central nervous system, neural networks and single neurons to reproduce the operating principles of neural processing. Special time-series statistical techniques (spike train analytical methods) are developed to analyze the signals encoded in the firing patterns of neurons with dynamical interactions. Experimental studies include *in vivo* neurophysiological experiments studying the motor control, sensori-motor integration and learning functions by recording the firing patterns of multiple neurons simultaneously from the motor and cerebellar cortices, and *in vitro* experiments studying neuronal network dynamics in signal processing and synaptic plasticity in learning and memory by recording the firing patterns of cultured neurons grown on a multi-microelectrode plate. It is an attempt to bridge the gap between theories and experiments in neuroscience by testing the hypotheses predicted in the models with specifically designed experiments.

RESEARCH PROJECTS

My research projects encompass in both experimental and theoretical neurophysiology. The current research efforts are devoted to the understanding of neural dynamics in neural networks and signal processing capabilities in network. The ultimate goal is to understand the parallel processing and ensemble encoding schemes used by networks of neurons. To accomplish this goal, this problem is tackled using multiple approaches (which are implemented in the multiple research projects):

(1) ***Experimental Approach***: Perform experiments to record from multiple (identifiable) neurons simultaneously *in vitro* and *in vivo* with specific physiological stimuli.

The *in vitro* project includes both cultured neuron preparation and, in near future, cultured slice (organotypic slice) preparation. The *in vitro* experiments involve co-culturing cerebellar cortical neurons with inferior olivary neurons on a multi-microelectrode plate for simultaneous multi-unit recording from Purkinje cells and electrical stimulation of olivary neurons. This project is aimed at uncovering principles of parallel signal processing in a network of neurons, the dynamics of a network and the emerging properties of cooperative firings in neural networks.

The *in vivo* project includes recording from behaving animals with implanted multi-electrodes in the cortical areas (cerebellum and motor cortex). This project is aimed at revealing principles for sensori-motor control and movement coordination by networks of neurons. The experimental paradigm for this study requires training rats to perform a motor task of bar pressing while recording from cerebellar cortical neurons and monitoring the associated movement dynamic and kinetic parameters of the animal. These experiments are aimed at acquiring neural data together with the associated physiological parameters for revealing the "physiologically correct" underlying mechanisms of neural signal processing and neural signal encoding in a network of neurons.

(2) ***Theoretical Approach***: Model network activity by a large-scale neural simulator for realistic modeling of biological neurons using well known biophysical and biochemical principles that are revealed by experiments.

Complex interactions among neurons are studied by computer simulation of large number of neurons to reveal phenomena may not be easily identified based on the properties of individual neurons. Based on these simulations, mathematical and algorithmic formularization of the underlying theories governing neural signal processing and signal encoding/decoding in the central and peripheral nervous systems can be extracted from the salient features exhibited from the simulation results.

(3) ***Spike Train Analysis Approach***: Analyze the signals generated by neurons either from experimental results or simulation results in the context of network activity and physiological functions.

Given the large number of multi-unit data recorded either from experimental preparations or from computer simulation, the next crucial step is to analyze these multi-unit data and interpret them with respect to the physiological functions of the animal. Multi-unit spike train analysis are one of the necessary tools for analyzing the neural network data quantitatively within the context of physiological functions and firing patterns in neurons. Several specially designed spike train statistical analysis of stochastic point processes are developed to reveal the spatio-temproal dynamics and interactions among multiple neurons based on the spike train signals.

PUBLICATIONS

Tam, D. C. (1992) Vectorial phase-space analysis for detecting dynamical interactions in firing patterns of biological neural networks. *Proceedings of the International Joint Conference on Neural Networks,* June 1992. Vol.3 pp.97-102.

Tam, D. C. (1992) Novel cross-interval maps for identifying attractors from multi-unit neural firing patterns. In: *Nonlinear Dynamical Analysis of the EEG..* (B. Jensen, ed.) World Scientific Publishing Co (in press)

Tam, D. C. (1992) A multi-neuronal vectoral phase-space analysis for detecting dynamical interactions in firing patterns of biological neural networks. In: Computational Neural Systems. (J. M. Bower, ed.) (in press)

Tam, D. C. and Hutson, R. K. (1992) An object-oriented paradigm for the design of realistic neural simulators. In: Computational Neural Systems. (J. M. Bower, ed.) (in press)

Tam, D. C. (1992) A novel vectorial phase-space analysis of spatio-temporal firing patterns in biological neural networks. *Proceedings of the Simulation Technology Conference.* (in press)

Tam, D. C. (1992) A generalizable object-oriented neural simulator for reconstructing functional properties of biological neuronal networks. *Proceedings of the Simulation Technology Conference.* (in press)

Tam, D. C. (1992) Object-oriented programming techniques for implementing generalizable models. *Proceedings of the Simulation Technology Conference.* (in press)

Tam, D. C. (1992) A hybrid time-shifted neural network for analyzing biological neuronal spike trains. *Progress in Neural Networks* (O. Omidvar, ed.) Vol. 2, Ablex Publishing Corporation: Norwood, New Jersey. (*in press*)

Tam, D. C. (1991) Signal processing in multi-threshold neurons. In: *Single Neuron Computation* (T. McKenna, J. Davis, and S. F. Zornetzer, eds.) Academic Press, San Diego. pp. 481-501.

Tam, D. C. (1991) Signal processing by multiplexing and demultiplexing in neurons. In: *Advances in Neural Information Processing Systems.* (D. S. Touretzky, ed.), Morgan Kaufmann Publishers, San Mateo, California. pp. 282-288.

Alkon, D. L., Vogl, T. P, Blackwell, K. T. and Tam, D. C. (1991) Memory function in neural and artificial networks. In: *Neural Network Models of Conditioning and Action.* (M. L. Commons, S. Grossberg, J. E. R. Staddon, eds.) pp. 1-11. Lawrence Erlbaum Associates: Hillsdale, New Jersey.

Tam, D. C. (1990) Decoding of firing intervals in a temporal-coded spike train using a topographically mapped neural network. *Proceedings of the International Joint Conference on Neural Networks, June, 1990.* Vol. 3, pp. III-627–632.

Tam, D. C. (1990) Temporal-spatial coding transformation: Conversion of frequency-code to place-code via a time-delayed neural network. *Proceedings of the International Joint Conference on Neural Networks* (H. Caudill, eds.), *Jan., 1990.* Vol. 1, pp. I-130–133.

Tam, D. C. (1989) The physiological basis and implications of differential motor activation. *Behavioral and Brain Sciences* Vol. 12, pp. 669.

Tam, D. C. (1989) A positive/negative reinforcement learning model for associative search network. *Proceedings of the First Annual IEEE Symposium on Parallel and Distributed Processing, 1989,* (B. Shirazi, ed.) pp. 300-307.

Tam, D. C. and Perkel, D. H. (1989) Quantitative modeiing of synaptic plasticity. In: *The Psychology of Learning and Motivation: Computational Models of Learning in Simple Neural Systems,* (R. D. Hawkins and G. H. Bower, eds.) Vol. 23, pp. 1 - 30. Academic Press: San Diego.

Tam, D. C. and Perkel, D. H. (1989) A model for temporal correlation of biological neuronal spike trains. *Proceedings of the IEEE International Joint Conference on Neural Networks 1989.* Vol. 1, pp. I-781–786.

Tam, D. C., Ebner, T. J., and Knox, C. K. (1988) Cross-interval histogram and cross-interspike interval histogram correlation analysis of simultaneously recorded multiple spike train data. *Journal of Neuroscience Methods,* Vol. 23, pp. 23-33.

ABSTRACTS

Tam, D. C. (1992) A new multi-conditional statistical measure for detecting spatio-temporally correlated firing patterns in multiple spike trains. *Society for Neuroscience Abstract.* (in press)

Kenyon, G. T. and Tam, D. C. (1992) Using the assymptotic kolmogorov entropy to identify deterministic structures in multi-unit spike trains. *Society for Neuroscience Abstract.* (in press)

Zouridakis, G. and Tam, D. C. (1992) Multi-unit spike discrimination using wavelet transforms. *Society for Neuroscience Abstract.* (in press)

Tam, D. C. and Kenyon, G. T. (1992) Novel Cross-Interval Maps for Identifying Attractors from Multi-Unit Neural Firing Patterns. *Second Annual Conference on Nonlinear Dynamical Analysis of the EEG.* (in press).

Tam, D. C. and Kenyon, G. T. (1992) A vectorial statistical method for detecting correlated firing patterns in neurons. *Biophysical Society Abstract.* Vol. 16, p.A175.

Kenyon, G. T. and Tam, D. C.(1992) An object-oriented paradigm for simulating physiological processes in extended biological structures. *Biophysical Society Abstract.* Vol. 16, p.A175.

Tam, D. C. and Kenyon, G. T. (1991) A novel vectorial measure for detecting temporally correlated firing patterns in multiple spike trains. *Society for Neuroscience Abstract.* Vol. 17, p. 125.

Boney, D. G., Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating inter-connected neural systems. *Society for Neuroscience Abstract.* Vol. 17, p. 126.

Tam, D. C. (1991) A vectorial statistical method for analyzing stochastic firing patterns in large numbers of neurons in parallel. *The Second Keck Symposium on Computational Biology Abstract.*

Feinswog, L. J., Hutson, R. K., Kenyon, G. T. and Tam, D. C. (1991) An object-oriented paradigm for simulating neurophysiological processes. *The Second Keck Symposium on Computational Biology Abstract.*

Tam, D. C. (1990) Functional significance of bi-threshold firing of neurons. *Society for Neuroscience Abstract.* Vol. 16, p. 1091.

Tam, D. C. (1990) Hebbian synapse and its relation to cross-correlation function in associative conditioning learning. *Eighth Annual Conference on Biomedical Engineering Research in Houston.* p. 5.

Tam, D. C. (1990) Visual-motor integration: What role does cerebellar cortical neurons participate in movement control in monkeys? *Eighth Annual Conference on Biomedical Engineering Research in Houston.* p. 39.

Tam, D. C. and McMullen, T. A. (1989) Hebbian synapses as cross-correlation functions in delay line circuitry *Society for Neuroscience Abstract* Vol. 15, p. 777.

Alkon, D. L., Vogl, T. P., Blackwell, K. T. and Tam, D. C. (1989) Pattern recognition and storage by an artificial network derived from biological systems. *Neural Network Models of Conditioning and Action: the Twelfth Symposium on Models of Behavior at Harvard University.*

Tam, D. C. and Perkel, D. H. (1988) Identification of firing patterns in multiple spike trains using a neural network employing the back-propagation error correction learning algorithm. *Society for Neuroscience Abstract*, Vol. 14, p. 260.

Ojakangas, C. L., Onstott, D. K., Tam, D. C., Ebner, T. J. (1988) Changes in hand kinematics during a quantifiable learning paradigm in primates. *Society for Neuroscience Abstract*, Vol. 24, p. 260.

Tam, D. C., Perkel, D. H., and Tucker, W. S. (1988) Correlation of multiple neuronal spike trains using the back-propagation error correction algorithm. *INNS 88 International Neural Network Society First Annual Meeting.*

Tam, D. C., Perkel, D. H., and Tucker, W. S. (1988) Temporal correlation of multiple neuronal spike trains using the back-propagation error correction algorithm. *INNS 88 International Neural Network Society First Annual Meeting.*

Tam, D. C. (1988) An improved reinforcement learning model for associative search network. *IEEE International Conference on Neural Network 1988.*

Tam, D. C., Ebner, T. J., and Knox, C. K. (1987) Correlation between Purkinje cell simple spike activity and movement kinematics during closed-loop, visually guided multi-joint movement with altered gain and delay feedback. *Society for Neuroscience Abstract*, Vol. 13, p.604.

Tam, D. C., Ebner, T. J., and Knox, C. K. (1986) Technique for evaluation of Purkinje cell activity during closed-loop visually guided movement requiring continual fine movement error correction. *Society for Neuroscience Abstract*, Vol. 12, p.1418.

Tam, D. C., Knox, C. K., and Ebner, T. J. (1985) Cross-interval correlation of firing pattern of simultaneously recorded neighboring cerebellar Purkinje cells. *Society for Neuroscience Abstract*, Vol. 11, p.1035.

Tam, D. C., Ebner, T. J., and Bloedel, J. R. (1982) The response properties of DSCT cells to periodic mechanical stimuli. *Society for Neuroscience Abstract*, Vol. 8, p. 957.

AN INTERSPIKE INTERVAL DECODING NEURAL NETWORK

THIS INVENTION WAS MADE WITH GOVERNMENT FUNDS;
THE GOVERNMENT HAS CERTAIN RIGHTS IN THE INVENTION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to multi-layered time-delayed neural networks useful in a variety of data and signal processing, image recognition and other computational tasks. In particular, the present invention relates to a means to convert serially encoded temporal firing intervals of a spike train waveform into a spatially distributed topographical matrix in which the interspike-interval and bandwidth information of the spike train may be extracted.

2. Description of the Related Technology

Topographical maps of neurons are found in the central nervous system of biological organisms for the tonotopical representation of tones, somatotopical representation of body surface, retinotopical representation of the visual world, etc. These topographical maps represent information based on the spatial locations of the neurons. The signals encoded by the biological neurons have characteristic pulse height and pulse width, and may be considered as pulse-coded signals where the information is encoded in the time of occurrence of the pulses.

This time-series of pulses is called a spike train and the neuron information is contained in the interspike-intervals between pulse firings from the neurons. Thus, the signal information transmitted by a neuron can be considered as "temporally-coded" by the time intervals between pulses in the spike train. Various methods of correlation analysis of spike trains in biological neurons have been developed. See for example "Neuronal spike trains and stochastic point process", Perkel, Gerstein and Moore, Biophys. J., Vol. 7, pp. 391-440 (1967); "Cross-interval histogram and cross-interspike interval histogram correlation analysis of simultaneously recorded multiple spike train data", Tam, Ebner and Knox, Journal of Neuroscience Methods, Vol. 23, pp. 23-33 (1967).

Given this serial transmission of the temporally-coded interspike pulse train, the information contained within the spike intervals may be decoded into parallel topographically distributed codes. Topographical distribution of codes based on the location of the neurons is called "place-code". By converting temporally-coded signals into topographically distributed codes, the firing intervals of neurons may be readily recognized as a particular neuron in a population ensemble. Thus, individual firing patterns may be distinguished such as burst-firing from long-interval firing and periodic firing from non-periodic firing.

Neural networks are characterized by a high degree of parallelism between numerous interconnected simple processors. The neural network nomenclature is derived from the similarity of such networks to biological neural networks. See for example "Computing with Neural Circuits: A Model", Hopfield and Tank, Vol. 233, pp. 622-33.

In general, neural networks are formed or modeled using a number of simple processors or neurons arranged in a highly interconnected pattern wherein each of the neurons performs the simple task of updating its output state based upon the value of signals presented to it as inputs. The design of a neural network involves determining the number, arrangement and weight of the interconnections between neurons. The weight of a connection corresponds to the biological synaptic strength and determines the degree in which the output signal on one neuron will effect the other neurons to which it is connected. Thus, each neuron or processor receives input signals which are derived from the output or activation states of other connected neurons.

These activation states or output signals are linearly, typically resistively, operated on via the connection weights and then summed. Summation may be accomplished in an analog circuit which adds together all input voltages to give a resultant voltage representative of the sum of the inputs. This input signal summation is then operated on by a non-linear processor function. such as a threshold detector, to produce an updated output activation state.

SUMMARY OF THE INVENTION

Information contained within the firing intervals of a temporally coded spike train may be decoded by a multi-layered neural network having some of its inputs delayed in time. This serially coded information can be converted into a spatially distributed two dimensional topographical map with multiple output channels each having decoded information serially presented for further processing. The system and method of this invention is well suited to perform the decoding, classifying and characterizing of the interspike-interval information and bandwidth of a serially encoded spike train.

An object of the present invention is to decode multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed topographically mapped individual channels.

Another object of the present invention is to extract the time variances of the incoming interspike intervals by distributing these variances on a two-dimensional output neuron array.

Yet another object of the present invention is to characterize the underlying stochastic processes of the incoming spike train firing intervals.

The foregoing objects are achieved as is now described. The system and method of this invention implements a signal processing scheme for code conversion using time delayed input and specifically connected neural networks for computing the interspike interval and bandwidth information embedded in a serial temporal spike train without requiring a learning mode.

The system and method of this invention processes a received input spike train comprised of a series of pulses having a spike time interval wherein the time value of the interval contains the information of interest. This spike train input is applied to an undelayed input of each of a plurality of first-layer neurons and first-parallel-layer neurons. The undelayed input of each the first neurons are effectively in parallel and receive the undelayed spike train input simultaneously.

The input spike train is also applied to a delay means that time shifts the input spike train so as to produce multiple cascaded replicas of the spike train delayed by multiples of incremental time intervals ("kΔt"). For example, Δt may be equal to one (1) millisecond ("msec") and k may represent values from 1 to n. Thus, time shifted replicas of the input spike train will begin 1 msec, 2 msec, 3 msec, 4 msec, . . . , and n msec after the original spike train. These time delayed versions of the spike train are systematically applied to inputs of the first-layer and first-parallel-layer neurons. Δt may be any time value so long as the spike interval time is greater than Δt.

Each first-layer and first-parallel-layer neurons are assigned a position in a network of the system of this invention. Each first neuron has a total number of inputs dependent on its position in the network. For example, the 1st neuron in the first layer has two inputs: one undelayed input connected directly to the spike train, and the other input connected to an output of the delay means whereby the spike train is delayed by Δt. The 2nd neuron has three inputs receiving the input spike train and cascaded time delayed replicas thereof: input (1) is undelayed, (2) is delayed by Δt and (3) is delayed by

2Δt. Similarly, the 3rd neuron has four inputs, the 4th neuron has five inputs, . . . and the nth neuron has n+1 inputs.

Both the first-layer and first-parallel-layer neurons may be thought of as a linear cascade of neurons having a progressively increasing number of inputs wherein these inputs receive successively increasing time delayed replicas of the input spike train. The purpose of both the first-layer and first-parallel-layer neurons are to indicate when there is a time coincidence of spikes at the inputs of each first neuron. Spike time coincidences at two or more inputs of the first-layer neurons produces an output. Similarly, spike time coincidences at three or more inputs of the first-parallel-layer neurons produces an output.

A first neuron may determine spike time and delayed spike time coincidence at its inputs by adding together all of the input signal voltages. Adding together the input voltages produces an internal voltage in the neuron representative of the sum of these input voltages. This internal summation voltage can be compared to a threshold voltage so as to perform a predefined logical operation.

For illustrative purposes assume that each first neuron input spike has a voltage amplitude value of one volt and the first-layer neurons have a threshold voltage of greater than one volt. Whenever two cr more inputs have spike voltages coincident in time, the internal neuron voltage sum will be greater than one volt and an output is produced. Thus the first-layer neurons perform a logical operation representative of two or more input spike voltages being coincident in time.

The first-parallel-layer neurons have a threshold voltage of greater than two volts. Similarly, when three or more inputs have spike voltages coincident in time, the internal neuron voltage sum will be greater than two volts and an output is produced. Thus the first-parallel-layer neurons perform a logical operation representative of three or more input spike voltages being coincident in time.

A single neuron with two threshold values and two outputs may perform the same function as a first-layer and a first-parallel-layer neuron combined. In addition, the first-layer neurons inhibit their outputs after an input spike time coincidence causes an output to occur. This output inhibition is for a time period of (k-1)Δt where k = 1 to n. For example, the 3rd first-layer neuron will not produce another output for (3-1)Δt or 2Δt. Using the above example 2Δt would equal 2 msec. This inhibitory time is called the refractory period. This refractory period is used to inhibit neuron outputs that would normally occur given the input spike time coincidence criteria mentioned above. A purpose of this refractory period is to compensate for the effects of phase differences between the original spike train and its time delayed replicas applied to the other inputs of the first-layer neurons.

Normally, an interspike interval will fall within a time delay window less than or equal to kΔt. For example, where k = 4 and Δt = 1 msec the spike interval must be less than or equal to 4 msec for the 4th or greater number first-layer neuron to fire (voltage signal on output). However, a spike interval of 2 msec will also cause the 4th or greater number first-layer neuron to fire. Thus, the first-layer neurons detect and fire on the first and higher order interspike intervals.

The first-parallel-layer neurons detect and fire on the second and higher order interspike intervals because its voltage threshold requires three on more inputs have time coincident spikes. Therefore by subtracting the outputs of the corresponding position first-parallel-layer neurons from the outputs of the corresponding position first-layer neurons, only the first order interspike intervals will be detected.

This subtraction process is performed in a plurality of second-layer neurons which eliminate higher order interspike interval detection. The outputs of the first-layer and first-parallel-layer neurons are applied to the inputs of the corresponding position second-layer neurons. Each second-layer neuron has two inputs: one excitatory and the other inhibitory. The excitatory second-layer neuron input receives the corresponding position first-layer neuron output and the inhibitory second-layer neuron input receives the corresponding position first-parallel-layer neuron output. Thus, for example, the 3rd first-layer neuron and the 3rd first-parallel-layer neuron outputs are connected to the 3rd second-layer neuron excitatory and inhibitory inputs respectively.

An output from the second-layer neurons occur when a corresponding position first-layer neuron output is applied to the excitatory input of the corresponding position second-layer neuron and there is no corresponding position first-parallel-layer neuron output applied to the inhibitory input of the second-layer neuron. Thus, if a first-parallel-layer output occurs in time coincidence with its corresponding position first-layer neuron output, then the corresponding position second-layer neuron output in inhibited even though the excitatory input of the second-layer neuron receives an output from the corresponding position first-layer neuron. This is how only the first order interspike intervals are detected. Thus, an accurate representation of true first order intervals with the time delay window is assured.

An interspike interval may be thought of as the reciprocal of instantaneous frequency. The time delay window, $k\Delta t$, may also be thought of as a high-pass filter where the interspike interval must be less than or equal in time to this window. Thus, this window sets the lowest instantaneous frequency that can be detected. Therefore, longer interspike intervals may be detected as the number of first neuron inputs and time delays (k) are increased.

Given the various high-pass filtered interspike intervals possible, a plurality of band-pass filtered intervals may be obtained by connecting a plurality of third-layer neurons, configured in a two dimensional matrix, to the second-layer neuron outputs. In addition, the third-layer neuron matrix also characterizes the bandwidth variations of these interspike intervals.

The third-layer neuron two dimensional matrix may consist of n columns and m rows (n,m) where the number of third-layer neurons is equal to the product of n and m. Each third-layer neuron has two inputs: one excitatory and the other inhibitory. The (k,h) third-layer neuron's excitatory input receives the k-th second-layer neuron output and its inhibitory input receives the h-th second-layer neuron output for $k = 1$ to n and $h = 1$ to m where $n > m$. In similar fashion to a second-layer neuron, the output of a third-layer neuron only will fire when its excitatory input has a spike voltage and its inhibitory input has no spike voltage.

As an example, let k = 6, h = 4 and Δt = 1 msec; thus, the longest detectable spike interval is 6 msec and the shortest detectable spike interval is 4 msec. Any spike interval greater than 6 msec. or less than 4 msec will not activate the output of the (6,4) third-layer neuron. Thus, any output from the (6,4) third-layer neuron will represent spike intervals from 4 to 6 msec with a bandwidth variance of 2 msec.

Arranging the third-layer neuron matrix so that interspike intervals are represented by a horizontal axis (h) and the bandwidth of the interspike interval variances are represented by the vertical axis (k) allows decoding of temporal codes to spatial codes and, in addition, may be used for the decoding of multiplexed signals. Since the received interval times and their bandwidth variances are readily discernible, each third-layer neuron output may be further processed as a discrete channel of specific information.

Uses for the third-layer neuron cutput information are in speech recognition, visual image recognition, etc. The system and method of this invention allows processing of complex neural type signal information similar to the type of nerve signal information found in animals and man. This invention has application, but is not limited to, image recognition of radar signals, video images, sonar signatures, speech patterns, and data compression and multiplexing as possibly used between the retinal and the central nervous system of a biological structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself; however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figs. 1 through 4 are schematic illustrations of the architecture of an interspike interval decoding neural network in accordance with the system and method of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The system and method of the present invention provides a novel neural network architecture in combination with a signal delay means that represents an implementation of a signal processing scheme for code conversion using time for computing and coding that does not require learning. This invention may be used to decode multiplexed pulse coded signals embedded serially in an incoming spike train into parallelly distributed topographically mapped demultiplexed channels having unique information represented in each channel.

The system and method of this invention comprises three layers of neurons: (1) a first-layer and first-parallel-layer, (2) a second-layer and (3) a third-layer arranged in a two dimensional matrix form. In addition a time delay means is used to time shift an incoming spike train signal so as to produce multiple cascaded time delayed replicas of this spike train.

Referring now to the drawings, and particularly to Fig. 1, the letters FL designates generally a first-layer neural network with both direct and time delayed signal inputs receiving an incoming spike train representative of multiplexed coded information. Network FL includes a plurality of processing units FL_1, FL_2, FL_3, . . ., FL_k. Each processing unit is designed to be suitable as a unit cell for integration in an analog VLSI integrated circuit. The function of each processing unit may also be simulated by software computer program means.

Generally, a spike train, x(t), may be defined as a time series of spikes (delta functions) with a total of n+1 spikes:

$$x(t) = \sum_{j=0}^{n} \delta(t-\tau_j)$$

Such that at time $t = \tau_j$, there is a spike occurring in the spike train given by the delta function, which satisfies:

$$\delta(t) = \begin{cases} 1, & t=0 \\ 0, & t\neq 0 \end{cases}$$

The time between spikes is the interspike-interval, I_j, and may be defined as the time interval between any two adjacent spikes

occurring at time τ_j and τ_{j-1}, where:

$$I_j = \tau_j - \tau_{j-1}, \text{ for } 0<j\leq n$$

Each processing unit, FL_i where $i = 1$ to k, has a first input 10 receiving the undelayed spike train signals and a second input 12 receiving a replica of the spike train delayed in time by Δt. As the processing unit number increases so does the number of its inputs receiving the progressively time delayed replicas of the spike train. For example, processing unit FL_2 also has a third input 14 receiving a replica of the spike train delayed in time by $2\Delta t$, thus unit FL_k has $k+1$ inputs with a k-th input 16 receiving a replica of the spike train delayed in time by $k\Delta t$.

Therefore, as illustrated in Fig. 1, the k-th neuron processing unit, FL_k, in the first layer has $k+1$ inputs, each input receiving a progressively time delayed replica of the spike train signal. Therefore, the input spike train is delayed by times given by $\tau = i\Delta t$, for $i = 0$ to k.

A feature of the system and method of this invention is the bandpass filtering of the interspike intervals by the first-layer neuron processors. If the sum of the inputs to the k-th first-layer neuron processor is defined as:

$$X_k(t) = \sum_{i=0}^{k} x(t-i\Delta t)$$

and the interspike interval of the original undelayed spike train falls within the time delay window, $I_j \leq k\Delta t$, for $0<j\leq n$, then the input sum may defined as:

$$X_k(t) = \sum_{i=0}^{k} x(t-i\Delta t)>1$$

Conversely, if $I_j > k\Delta t$, for $0 < j \leq n$, then $X_k(t) \leq 1$. Thus, if the threshold for the input sum is set at greater than one, then the k-th neuron processor output will fire only when the interspike interval, I_j, of the input spike train is within the time delay window, $k\Delta t$. The output of the k-th first-layer neuron processor, FL_k, may be represented by:

$$y_k(t) = \begin{cases} 1, & \text{if } X_k > 1 \\ 0, & \text{otherwise} \end{cases}$$

Therefore, the k-th neuron first-layer processor may be considered to encode a bandpass filtered input interspike interval, $0 < I_j \leq k\Delta t$, wherein the k-th neuron output 20 will fire only if the original input spike train contains an interspike interval below this cutoff interval of $k\Delta t$.

In order to ensure that a first-layer neuron processor output will fire with a pulse of duration Δt only, given the various phase differences of the incoming delayed replicas of the input spike train, a refractory period of $(k-1)\Delta t$ for the k-th neuron is used. The output of the k-th neuron is inhibited, even if the input criteria are met, during the refractory time period.

If more than two spikes occur within the time delay window, $\tau = k\Delta t$, then the first-layer neuron processor may be overestimating the cutoff interspike interval, in which case the interspike intervals are shorter than the cutoff interval, $k\Delta t$. To prevent the first-layer neurons from estimating higher order interspike intervals instead of the first order interspike intervals, another set of neurons parallel to this first-layer is added and connected in a similar fashion.

These parallel neurons are called first-parallel-layer neurons and have their inputs connected in the same fashion as do the first-layer neuron inputs.

Referring now to Fig. 2, first-parallel-layer neuron processors, FPL_i where i = 1 to k, function in the same way as do the first-layer neuron processors except that the input summation threshold is set at greater than two instead of greater than one. The output of the k-th first-parallel-layer neuron processor, FPL_k, may be represented by:

$$y'_k(t) = \begin{cases} 1, \text{ if } X_k>2 \\ 0, \text{ otherwise} \end{cases}$$

Since the first-layer neuron processors detect first and higher order interspike intervals and the first-parallel-layer neuron processors detect second and higher order interspike intervals, taking the difference between these two sets of neuron processors results in the detection of only the first order interspike intervals. The system and method of this invention obtains the difference between the first-layer and first-parallel-layer neuron outputs using second-layer neuron processors connected to the respectively positioned first neuron processor outputs.

Referring now to Fig. 3, second-layer neuron processors, SL_i where i = 1 to k, each have an excitatory input 30 connected to the respective position first-layer neuron output 20 and an inhibitory input 32 connected to the respective position first-parallel-layer neuron output 22. The output of the k-th second-layer neuron processor, SL_k, may be represented by:

$$y''_k(t) = y_k(t)-y'_k(t) = \begin{cases} 1, \text{ if } 2\geq X_k(t)>1 \\ 0, \text{ if otherwise} \end{cases}$$

This ensures an accurate estimation of the first order interspike interval, $0<I_j\leq k\Delta t$, within the time delay window $k\Delta t$.

Given that the k-th neuron processor output in the second-layer indicates the first order interspike intervals represented by $0<I_j\leq k\Delta t$, and the h-th second-layer neuron processor output indicates the first order interspike intervals represented by $0<I_j\leq h\Delta t$, then the difference between the k-th and h-th second-layer neuron processor outputs represents the first order interspike intervals with a bandwidth of $h\Delta t$, i.e., $(k-h)\Delta t\leq I_j\leq k\Delta t$.

Referring now to Fig. 4, third-layer neuron processors $TL_{k,h}$, where $k = 1$ to n, $h = 1$ to m and $n > m$, are arranged in a two dimensional matrix of n columns and m rows (n,m). Each third-layer neuron processor has an excitatory input 50 connected tc the output of the second-layer neuron located at a position corresponding to the third-layer neuron column position and an inhibitory input 52 connected to the output of the second-layer neuron located at a position corresponding to the third-layer neuron row position. As illustrated in Fig. 4, each second-layer neuron output may be connected to a plurality of third-layer neuron inputs. As may be noted, only the third-layer neurons whose matrix column number k is greater than row number h are connected to the corresponding position second-layer neurons.

The output of the (k,h) third-layer neuron processor, $TL_{k,h}$, may be represented by:

$$y'''_{kh}(t)=y''_{k}(t)-y''_{h}(t)=\begin{cases}1, & \text{if } 2\geq\sum_{i=k-h}^{k}x(t-i\Delta t)>1\\ 0, & \text{otherwise}\end{cases}$$

Thus, the third-layer neuron processor located at the k-th column and h-th row receives excitatory input from the k-th second-layer neuron output and inhibitory input from the h-th second-layer neuron output. The (k,h) position third-layer neuron processor indicates interspike intervals within the range of (k-h)Δt and kΔt when there is an output from the k-th second-layer neuron and an absence of an output from the h-th second-layer neuron.

The system and method of this invention may be used to detect both the exact interspike intervals and intervals with some variance. The interspike interval accuracy and acceptable tolerance may be selected from the topographical location of third-layer neuron processors. Thus, the interspike intervals of the input spike train may be sorted into a distributed set of bandpass filtered spike trains whose topological position within the third-layer neuron processor output matrix indicates the various interspike intervals and their associated bandwidth intervals.

The system of the present invention, therefore, is well adapted to carry out the objects and attain the ends and advantages mentioned as well as others inherent therein. While a presently preferred embodiment of the invention has been given for the purpose of disclosure, numerous changes in the details of construction and arrangement of parts will readily suggest themselves to those skilled in the art and which are encompassed within the spirit of the invention and the scope of the appended claims.

CLAIMS:

What is claimed is:

1. A neural network interspike interval decoding system for use in decoding multiplexed pulse-coded signals embedded serially in an incoming spike train waveform into parallel distributed topographically mapped channels, said network being comprised of a plurality of interconnected neurons wherein said network comprises:

means for time delay of the incoming spike train by time intervals of Δt, said delay means having a plurality of outputs providing replicas of the incoming spike train delayed in time by kΔt for k = 1 to n;

n first-layer neurons, each of said first-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delay means outputs for k = 1 to n;

each of said first-layer neurons having an output means providing a first-layer neuron output pulse of time duration Δt when at least two of said corresponding first-layer neuron inputs have spikes coincident in time and no previous output pulse has occurred within a time of (k-1)Δt,

n first-parallel-layer neurons, each of said first-parallel-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delay means outputs for k = 1 to n;

each of said first-parallel-layer neurons having an output means providing a first-parallel-layer neuron output pulse when at least three of said corresponding first-parallel-layer neuron inputs have spikes coincident in time;

n second-layer neurons, each of said second-layer neurons having an excitatory input in communication with said corresponding position first-layer neuron output and an inhibitory input in communication with said corresponding position first-parallel-layer neuron output;

each of said second-layer neurons having an output means providing a second-layer neuron output pulse only when there is a pulse on said excitatory input and an absence of a pulse on said inhibitory input;

n X m third-layer neurons arranged in a matrix of n columns and m rows (n,m), each of said third-layer (k,h) neurons having an excitatory input connected to said corresponding position k second-layer neuron output and an inhibitory input connected to said corresponding h position second-layer neuron output for k = 1 to n and h = 1 to m, where n > m; and

each of said third-layer neurons having an output means providing a third-layer neuron output pulse at matrix location (k,h) only when there is a pulse on said k column third-layer excitatory input and an absence of a pulse on said h row third-layer inhibitory input for k = 1 to n and h = 1 to m, where n > m.

2. A method for creating a neural network interspike interval decoding system for use in decoding multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed

topographically mapped channels, said network being comprised of a plurality of interconnected neurons having n first-layer neurons each with one undelayed spike train input and k time delayed inputs, n first-parallel-layer neurons each with one undelayed spike train input and k time delayed inputs, n second-layer neurons and an n X m third-layer neuron matrix interconnected to decode the firing interspike-intervals and bandwidth variations of the spike train, said method comprising the steps of:

delaying in time the incoming spike train by time intervals of kΔt for k = 1 to n;

establishing n first-layer neurons, each of said first-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delayed spike train for k = 1 to n;

generating a first-layer neuron output pulse of time duration Δt when at least two of said corresponding first-layer neuron inputs have spikes coincident in time and no previous output pulse has occurred within a time of (k-1)Δt,

establishing n first-parallel-layer neurons, each of said first-parallel-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delayed spike train for k = 1 to n;

generating a first-parallel-layer neuron output pulse when at least three of said corresponding first-parallel-layer neuron inputs have spikes coincident in time;

establishing n second-layer neurons, each of said second-layer neurons having an excitatory inp.t in communication with said corresponding position first-layer neuron output and an inhibitory input in communication with said corresponding position first-parallel-layer neuron output;

generating a second-layer neuron output pulse only when there is a pulse on said excitatory input and an absence of a pulse on said inhibitory input;

establishing n X m third-layer neurons arranged in a matrix of n columns and m rows (n,m), each of said third-layer (k,h) neurons having an excitatory input connected to said corresponding position k second-layer neuron output and an inhibitory input connected to said corresponding h position second-layer neuron output for k = 1 to n and h = 1 to m, where n > m; and

generating a third-layer neuron output pulse at matrix location (k,h) only when there is a pulse on said k column third-layer excitatory input and an absence of a pulse on said h row third-layer inhibitory input for k = 1 to n and h = 1 to m, where n > m.

ABSTRACT OF THE DISCLOSURE

A multi-layered neural network is disclosed that converts an incoming temporally coded spike train into a spatially distributed topographical map from which interspike-interval and bandwidth information may be extracted. This neural network may be used to decode multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed topographically mapped channels. A signal processing and code conversion algorithm not requiring learning is provided.

AN INTERSPIKE INTERVAL DECODING NEURAL NETWORK

THIS INVENTION WAS MADE WITH GOVERNMENT FUNDS;
THE GOVERNMENT HAS CERTAIN RIGHTS IN THE INVENTION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to multi-layered time-delayed neural networks useful in a variety of data and signal processing, image recognition and other computational tasks. In particular, the present invention relates to a means to convert serially encoded temporal firing intervals of a spike train waveform into a spatially distributed topographical matrix in which the interspike-interval and bandwidth information of the spike train may be extracted.

2. Description of the Related Technology

Topographical maps of neurons are found in the central nervous system of biological organisms for the tonotopical representation of tones, somatotopical representation of body surface, retinotopical representation of the visual world, etc. These topographical maps represent information based on the spatial locations of the neurons. The signals encoded by the biological neurons have characteristic pulse height and pulse width, and may be considered as pulse-coded signals where the information is encoded in the time of occurrence of the pulses.

This time-series of pulses is called a spike train and the neuron information is contained in the interspike-intervals between pulse firings from the neurons. Thus, the signal information transmitted by a neuron can be considered as "temporally-coded" by the time intervals between pulses in the spike train. Various methods of correlation analysis of spike trains in biological neurons have been developed. See for example "Neuronal spike trains and stochastic point process", Perkel, Gerstein and Moore, Biophys. J., Vol. 7, pp. 391-440 (1967); "Cross-interval histogram and cross-interspike interval histogram correlation analysis of simultaneously recorded multiple spike train data", Tam, Ebner and Knox, Journal of Neuroscience Methods, Vol. 23, pp. 23-33 (1967).

Given this serial transmission of the temporally-coded interspike pulse train, the information contained within the spike intervals may be decoded into parallel topographically distributed codes. Topographical distribution of codes based on the location of the neurons is called "place-code". By converting temporally-coded signals into topographically distributed codes, the firing intervals of neurons may be readily recognized as a particular neuron in a population ensemble. Thus, individual firing patterns may be distinguished such as burst-firing from long-interval firing and periodic firing from non-periodic firing.

Neural networks are characterized by a high degree of parallelism between numerous interconnected simple processors. The neural network nomenclature is derived from the similarity of such networks to biological neural networks. See for example "Computing with Neural Circuits: A Model", Hopfield and Tank, Vol. 233, pp. 622-33.

In general, neural networks are formed or modeled using a number of simple processors or neurons arranged in a highly interconnected pattern wherein each of the neurons performs the simple task of updating its output state based upon the value of signals presented to it as inputs. The design of a neural network involves determining the number, arrangement and weight of the interconnections between neurons. The weight of a connection corresponds to the biological synaptic strength and determines the degree in which the output signal on one neuron will effect the other neurons to which it is connected. Thus, each neuron or processor receives input signals which are derived from the output or activation states of other connected neurons.

These activation states or output signals are linearly, typically resistively, operated on via the connection weights and then summed. Summation may be accomplished in an analog circuit which adds together all input voltages to give a resultant voltage representative of the sum of the inputs. This input signal summation is then operated on by a non-linear processor function. such as a threshold detector, to produce an updated output activation state.

SUMMARY OF THE INVENTION

Information contained within the firing intervals of a temporally coded spike train may be decoded by a multi-layered neural network having some of its inputs delayed in time. This serially coded information can be converted into a spatially distributed two dimensional topographical map with multiple output channels each having decoded information serially presented for further processing. The system and method of this invention is well suited to perform the decoding, classifying and characterizing of the interspike-interval information and bandwidth of a serially encoded spike train.

An object of the present invention is to decode multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed topographically mapped individual channels.

Another object of the present invention is to extract the time variances of the incoming interspike intervals by distributing these variances on a two-dimensional output neuron array.

Yet another object of the present invention is to characterize the underlying stochastic processes of the incoming spike train firing intervals.

The foregoing objects are achieved as is now described. The system and method of this invention implements a signal processing scheme for code conversion using time delayed input and specifically connected neural networks for computing the interspike interval and bandwidth information embedded in a serial temporal spike train without requiring a learning mode.

The system and method of this invention processes a received input spike train comprised of a series of pulses having a spike time interval wherein the time value of the interval contains the information of interest. This spike train input is applied to an undelayed input of each of a plurality of first-layer neurons and first-parallel-layer neurons. The undelayed input of each the first neurons are effectively in parallel and receive the undelayed spike train input simultaneously.

The input spike train is also applied to a delay means that time shifts the input spike train so as to produce multiple cascaded replicas of the spike train delayed by multiples of incremental time intervals ("kΔt"). For example, Δt may be equal to one (1) millisecond ("msec") and k may represent values from 1 to n. Thus, time shifted replicas of the input spike train will begin 1 msec, 2 msec, 3 msec, 4 msec, . . . , and n msec after the original spike train. These time delayed versions of the spike train are systematically applied to inputs of the first-layer and first-parallel-layer neurons. Δt may be any time value so long as the spike interval time is greater than Δt.

Each first-layer and first-parallel-layer neurons are assigned a position in a network of the system of this invention. Each first neuron has a total number of inputs dependent on its position in the network. For example, the 1st neuron in the first layer has two inputs: one undelayed input connected directly to the spike train, and the other input connected to an output of the delay means whereby the spike train is delayed by Δt. The 2nd neuron has three inputs receiving the input spike train and cascaded time delayed replicas thereof: input (1) is undelayed, (2) is delayed by Δt and (3) is delayed by

2Δt. Similarly, the 3rd neuron has four inputs, the 4th neuron has five inputs, . . . and the nth neuron has n+1 inputs.

Both the first-layer and first-parallel-layer neurons may be thought of as a linear cascade of neurons having a progressively increasing number of inputs wherein these inputs receive successively increasing time delayed replicas of the input spike train. The purpose of both the first-layer and first-parallel-layer neurons are to indicate when there is a time coincidence of spikes at the inputs of each first neuron. Spike time coincidences at two or more inputs of the first-layer neurons produces an output. Similarly, spike time coincidences at three or more inputs of the first-parallel-layer neurons produces an output.

A first neuron may determine spike time and delayed spike time coincidence at its inputs by adding together all of the input signal voltages. Adding together the input voltages produces an internal voltage in the neuron representative of the sum of these input voltages. This internal summation voltage can be compared to a threshold voltage so as to perform a predefined logical operation.

For illustrative purposes assume that each first neuron input spike has a voltage amplitude value of one volt and the first-layer neurons have a threshold voltage of greater than one volt. Whenever two or more inputs have spike voltages coincident in time, the internal neuron voltage sum will be greater than one volt and an output is produced. Thus the first-layer neurons perform a logical operation representative of two or more input spike voltages being coincident in time.

The first-parallel-layer neurons have a threshold voltage of greater than two volts. Similarly, when three or more inputs have spike voltages coincident in time, the internal neuron voltage sum will be greater than two volts and an output is produced. Thus the first-parallel-layer neurons perform a logical operation representative of three or more input spike voltages being coincident in time.

A single neuron with two threshold values and two outputs may perform the same function as a first-layer and a first-parallel-layer neuron combined. In addition, the first-layer neurons inhibit their outputs after an input spike time coincidence causes an output to occur. This output inhibition is for a time period of $(k-1)\Delta t$ where $k = 1$ to n. For example, the 3rd first-layer neuron will not produce another output for $(3-1)\Delta t$ or $2\Delta t$. Using the above example $2\Delta t$ would equal 2 msec. This inhibitory time is called the refractory period. This refractory period is used to inhibit neuron outputs that would normally occur given the input spike time coincidence criteria mentioned above. A purpose of this refractory period is to compensate for the effects of phase differences between the original spike train and its time delayed replicas applied to the other inputs of the first-layer neurons.

Normally, an interspike interval will fall within a time delay window less than or equal to $k\Delta t$. For example, where $k = 4$ and $\Delta t = 1$ msec the spike interval must be less than or equal to 4 msec for the 4th or greater number first-layer neuron to fire (voltage signal on output). However, a spike interval of 2 msec will also cause the 4th or greater number first-layer neuron to fire. Thus, the first-layer neurons detect and fire on the first and higher order interspike intervals.

The first-parallel-layer neurons detect and fire on the second and higher order interspike intervals because its voltage threshold requires three on more inputs have time coincident spikes. Therefore by subtracting the outputs of the corresponding position first-parallel-layer neurons from the outputs of the corresponding position first-layer neurons, only the first order interspike intervals will be detected.

This subtraction process is performed in a plurality of second-layer neurons which eliminate higher order interspike interval detectior. The outputs of the first-layer and first-parallel-layer neurons are applied to the inputs of the corresponding position second-layer neurons. Each second-layer neuron has two inputs: one excitatory and the other inhibitory. The excitatory second-layer neuron input receives the corresponding position first-layer neuron output and the inhibitory second-layer neuron input receives the corresponding position first-parallel-layer neuron output. Thus, for example, the 3rd first-layer neuron and the 3rd first-parallel-layer neuron outputs are connected to the 3rd second-layer neuron excitatory and inhibitory inputs respectively.

An output from the second-layer neurons occur when a corresponding position first-layer neuron output is applied to the excitatory input of the corresponding position second-layer neuron and there is no corresponding position first-parallel-layer neuron output applied to the inhibitory input of the second-layer neuron. Thus, if a first-parallel-layer output occurs in time coincidence with its corresponding position first-layer neuron output, then the corresponding position second-layer neuron output in inhibited even though the excitatory input of the second-layer neuron receives an output from the corresponding position first-layer neuron. This is how only the first order interspike intervals are detected. Thus, an accurate representation of true first order intervals with the time delay window is assured.

An interspike interval may be thought of as the reciprocal of instantaneous frequency. The time delay window, $k\Delta t$, may also be thought of as a high-pass filter where the interspike interval must be less than or equal in time to this window. Thus, this window sets the lowest instantaneous frequency that can be detected. Therefore, longer interspike intervals may be detected as the number of first neuron inputs and time delays (k) are increased.

Given the various high-pass filtered interspike intervals possible, a plurality of band-pass filtered intervals may be obtained by connecting a plurality of third-layer neurons, configured in a two dimensional matrix, to the second-layer neuron outputs. In addition, the third-layer neuron matrix also characterizes the bandwidth variations of these interspike intervals.

The third-layer neuron two dimensional matrix may consist of n columns and m rows (n,m) where the number of third-layer neurons is equal to the product of n and m. Each third-layer neuron has two inputs: one excitatory and the other inhibitory. The (k,h) third-layer neuron's excitatory input receives the k-th second-layer neuron output and its inhibitory input receives the h-th second-layer neuron output for $k = 1$ to n and $h = 1$ to m where $n > m$. In similar fashion to a second-layer neuron, the output of a third-layer neuron only will fire when its excitatory input has a spike voltage and its inhibitory input has no spike voltage.

As an example, let k = 6, h = 4 and Δt = 1 msec; thus, the longest detectable spike interval is 6 msec and the shortest detectable spike interval is 4 msec. Any spike interval greater than 6 msec. or less than 4 msec will not activate the output of the (6,4) third-layer neuron. Thus, any output from the (6,4) third-layer neuron will represent spike intervals from 4 to 6 msec with a bandwidth variance of 2 msec.

Arranging the third-layer neuron matrix so that interspike intervals are represented by a horizontal axis (h) and the bandwidth of the interspike interval variances are represented by the vertical axis (k) allows decoding of temporal codes to spatial codes and, in addition, may be used for the decoding of multiplexed signals. Since the received interval times and their bandwidth variances are readily discernible, each third-layer neuron output may be further processed as a discrete channel of specific information.

Uses for the third-layer neuron output information are in speech recognition, visual image recognition, etc. The system and method of this invention allows processing of complex neural type signal information similar to the type of nerve signal information found in animals and man. This invention has application, but is not limited to, image recognition of radar signals, video images, sonar signatures, speech patterns, and data compression and multiplexing as possibly used between the retinal and the central nervous system of a biological structure.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself; however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figs. 1 through 4 are schematic illustrations of the architecture of an interspike interval decoding neural network in accordance with the system and method of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The system and method of the present invention provides a novel neural network architecture in combination with a signal delay means that represents an implementation of a signal processing scheme for code conversion using time for computing and coding that does not require learning. This invention may be used to decode multiplexed pulse coded signals embedded serially in an incoming spike train into parallelly distributed topographically mapped demultiplexed channels having unique information represented in each channel.

The system and method of this invention comprises three layers of neurons: (1) a first-layer and first-parallel-layer, (2) a second-layer and (3) a third-layer arranged in a two dimensional matrix form. In addition a time delay means is used to time shift an incoming spike train signal so as to produce multiple cascaded time delayed replicas of this spike train.

Referring now to the drawings, and particularly to Fig. 1, the letters FL designates generally a first-layer neural network with both direct and time delayed signal inputs receiving an incoming spike train representative of multiplexed coded information. Network FL includes a plurality of processing units FL_1, FL_2, FL_3, . . ., FL_k. Each processing unit is designed to be suitable as a unit cell for integration in an analog VLSI integrated circuit. The function of each processing unit may also be simulated by software computer program means.

Generally, a spike train, x(t), may be defined as a time series of spikes (delta functions) with a total of n+1 spikes:

$$x(t) = \sum_{j=0}^{n} \delta(t-\tau_j)$$

Such that at time $t = \tau_j$, there is a spike occurring in the spike train given by the delta function, which satisfies:

$$\delta(t) = \begin{cases} 1, & t=0 \\ 0, & t=0 \end{cases}$$

The time between spikes is the interspike-interval, I_j, and may be defined as the time interval between any two adjacent spikes
occurring at time τ_j and τ_{j-1}, where:

$$I_j = \tau_j - \tau_{j-1}, \text{ for } 0<j<n$$

Each processing unit, FL_i where $i = 1$ to k, has a first input 10 receiving the undelayed spike train signals and a second input 12 receiving a replica of the spike train delayed in time by Δt. As the processing unit number increases so does the number of its inputs receiving the progressively time delayed replicas of the spike train. For example, processing unit FL_2 also has a third input 14 receiving a replica of the spike train delayed in time by $2\Delta t$, thus unit FL_k has $k+1$ inputs with a k-th input 16 receiving a replica of the spike train delayed in time by $k\Delta t$.

Therefore, as illustrated in Fig. 1, the k-th neuron processing unit, FL_k, in the first layer has $k+1$ inputs, each input receiving a progressively time delayed replica of the spike train signal. Therefore, the input spike train is delayed by times given by $\tau = i\Delta t$, for $i = 0$ to k.

A feature of the system and method of this invention is the bandpass filtering of the interspike intervals by the first-layer neuron processors. If the sum of the inputs to the k-th first-layer neuron processor is defined as:

$$X_k(t) = \sum_{i=0}^{k} x(t-i\Delta t)$$

and the interspike interval of the original undelayed spike train falls within the time delay window, $I_j<k\Delta t$, for $0<j<n$, then the input sum may defined as:

$$X_k(t) = \sum_{i=0}^{k} x(t-i\Delta t)>1$$

Conversely, if $I_j > k\Delta t$, for $0<j<n$, then $X_k(t)<1$. Thus, if the threshold for the input sum is set at greater than one, then the k-th neuron processor output will fire only when the interspike interval, I_j, of the input spike train is within the time delay window, $k\Delta t$. The output of the k-th first-layer neuron processor, FL_k, may be represented by:

$$y_k(t) = \begin{cases} 1, & \text{if } X_k > 1 \\ 0, & \text{otherwise} \end{cases}$$

Therefore, the k-th neuron first-layer processor may be considered to encode a bandpass filtered input interspike interval, $0<I_j<k\Delta t$, wherein the k-th neuron output 20 will fire only if the original input spike train contains an interspike interval below this cutoff interval of $k\Delta t$.

In order to ensure that a first-layer neuron processor output will fire with a pulse of duration Δt only, given the various phase differences of the incoming delayed replicas of the input spike train, a refractory period of $(k-1)\Delta t$ for the k-th neuron is used. The output of the k-th neuron is inhibited, even if the input criteria are met, during the refractory time period.

If more than two spikes occur within the time delay window, $\tau = k\Delta t$, then the first-layer neuron processor may be overestimating the cutoff interspike interval, in which case the interspike intervals are shorter than the cutoff interval, $k\Delta t$. To prevent the first-layer neurons from estimating higher order interspike intervals instead of the first order interspike intervals, another set of neurons parallel to this first-layer is added and connected in a similar fashion.

These parallel neurons are called first-parallel-layer neurons and have their inputs connected in the same fashion as do the first-layer neuron inputs.

Referring now to Fig. 2, first-parallel-layer neuron processors, FPL_i where i = 1 to k, function in the same way as do the first-layer neuron processors except that the input summation threshold is set at greater than two instead of greater than one. The output of the k-th first-parallel-layer neuron processor, FPL_k, may be represented by:

$$y'_k(t) = \begin{cases} 1, \text{ if } X_k>2 \\ 0, \text{ otherwise} \end{cases}$$

Since the first-layer neuron processors detect first and higher order interspike intervals and the first-parallel-layer neuron processors detect second and higher order interspike intervals, taking the difference between these two sets of neuron processors results in the detection of only the first order interspike intervals. The system and method of this invention obtains the difference between the first-layer and first-parallel-layer neuron outputs using second-layer neuron processors connected to the respectively positioned first neuron processor outputs.

Referring now to Fig. 3, second-layer neuron processors, SL_i where i = 1 to k, each have an excitatory input 30 connected to the respective position first-layer neuron output 20 and an inhibitory input 32 connected to the respective position first-parallel-layer neuron output 22. The output of the k-th second-layer neuron processor, SL_k, may be represented by:

$$y''_k(t) = y_k(t)-y'_k(t) = \begin{cases} 1, \text{ if } 2>X_k(t)>1 \\ 0, \text{ if otherwise} \end{cases}$$

This ensures an accurate estimation of the first order interspike interval, $0<I_j<k\Delta t$, within the time delay window $k\Delta t$.

Given that the k-th neuron processor output in the second-layer indicates the first order interspike intervals represented by $0<I_j<k\Delta t$, and the h-th second-layer neuron processor output indicates the first order interspike intervals represented by $0<I_j<h\Delta t$, then the difference between the k-th and h-th second-layer neuron processor outputs represents the first order interspike intervals with a bandwidth of $h\Delta t$, i.e., $(k-h)\Delta t<I_j<k\Delta t$.

Referring now to Fig. 4, third-layer neuron processors $TL_{k,h}$, where $k = 1$ to n, $h = 1$ to m and $n > m$, are arranged in a two dimensional matrix of n columns and m rows (n,m). Each third-layer neuron processor has an excitatory input 50 connected to the output of the second-layer neuron located at a position corresponding to the third-layer neuron column position and an inhibitory input 52 connected to the output of the second-layer neuron located at a position corresponding to the third-layer neuron row position. As illustrated in Fig. 4, each second-layer neuron output may be connected to a plurality of third-layer neuron inputs. As may be noted, only the third-layer neurons whose matrix column number k is greater than row number h are connected to the corresponding position second-layer neurons.

The output of the (k,h) third-layer neuron processor, $TL_{k,h}$, may be represented by:

$$y'''_{kh}(t)=y''_{k}(t)-y''_{h}(t)=\begin{cases} 1, \text{ if } 2>\sum\limits_{i=k-h}^{k} x(t-i\Delta t)>1 \\ 0, \text{ otherwise} \end{cases}$$

Thus, the third-layer neuron processor located at the k-th column and h-th row receives excitatory input from the k-th second-layer neuron output and inhibitory input from the h-th second-layer neuron output. The (k,h) position third-layer neuron processor indicates interspike intervals within the range of (k-h)Δt and kΔt when there is an output from the k-th second-layer neuron and an absence of an output from the h-th second-layer neuron.

The system and method of this invention may be used to detect both the exact interspike intervals and intervals with some variance. The interspike interval accuracy and acceptable tolerance may be selected from the topographical location of third-layer neuron processors. Thus, the interspike intervals of the input spike train may be sorted into a distributed set of bandpass filtered spike trains whose topological position within the third-layer neuron processor output matrix indicates the various interspike intervals and their associated bandwidth intervals.

The system of the present invention, therefore, is well adapted to carry out the objects and attain the ends and advantages mentioned as well as others inherent therein. While a presently preferred embodiment of the invention has been given for the purpose of disclosure, numerous changes in the details of construction and arrangement of parts will readily suggest themselves to those skilled in the art and which are encompassed within the spirit of the invention and the scope of the appended claims.

CLAIMS:

What is claimed is:

1. A neural network interspike interval decoding system for use in decoding multiplexed pulse-coded signals embedded serially in an incoming spike train waveform into parallel distributed topographically mapped channels, said network being comprised of a plurality of interconnected neurons wherein said network comprises:

means for time delay of the incoming spike train by time intervals of Δt, said delay means having a plurality of outputs providing replicas of the incoming spike train delayed in time by kΔt for k = 1 to n;

n first-layer neurons, each of said first-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delay means outputs for k = 1 to n;

each of said first-layer neurons having an output means providing a first-layer neuron output pulse of time duration Δt when at least two of said corresponding first-layer neuron inputs have spikes coincident in time and no previous output pulse has occurred within a time of (k-1)Δt,

n first-parallel-layer neurons, each of said first-parallel-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delay means outputs for k = 1 to n;

each of said first-parallel-layer neurons having an output means providing a first-parallel-layer neuron output pulse when at least three of said corresponding first-parallel-layer neuron inputs have spikes coincident in time;

n second-layer neurons, each of said second-layer neurons having an excitatory input in communication with said corresponding position first-layer neuron output and an inhibitory input in communication with said corresponding position first-parallel-layer neuron output;

each of said second-layer neurons having an output means providing a second-layer neuron output pulse only when there is a pulse on said excitatory input and an absence of a pulse on said inhibitory input;

n X m third-layer neurons arranged in a matrix of n columns and m rows (n,m), each of said third-layer (k,h) neurons having an excitatory input connected to said corresponding position k second-layer neuron output and an inhibitory input connected to said corresponding h position second-layer neuron output for k = 1 to n and h = 1 to m, where n > m; and

each of said third-layer neurons having an output means providing a third-layer neuron output pulse at matrix location (k,h) only when there is a pulse on said k column third-layer excitatory input and an absence of a pulse on said h row third-layer inhibitory input for k = 1 to n and h = 1 to m, where n > m.

2. A method for creating a neural network interspike interval decoding system for use in decoding multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed

topographically mapped channels, said network being comprised of a plurality of interconnected neurons having n first-layer neurons each with one undelayed spike train input and k time delayed inputs, n first-parallel-layer neurons each with one undelayed spike train input and k time delayed inputs, n second-layer neurons and an n X m third-layer neuron matrix interconnected to decode the firing interspike-intervals and bandwidth variations of the spike train, said method comprising the steps of:

delaying in time the incoming spike train by time intervals of kΔt for k = 1 to n;

establishing n first-layer neurons, each of said first-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delayed spike train for k = 1 to n;

generating a first-layer neuron output pulse of time duration Δt when at least two of said corresponding first-layer neuron inputs have spikes coincident in time and no previous output pulse has occurred within a time of (k-1)Δt,

establishing n first-parallel-layer neurons, each of said first-parallel-layer neurons having one input connected to the undelayed incoming spike train and k inputs connected to said corresponding time delayed spike train for k = 1 to n;

generating a first-parallel-layer neuron output pulse when at least three of said corresponding first-parallel-layer neuron inputs have spikes coincident in time;

establishing n second-layer neurons, each of said second-layer neurons having an excitatory input in communication with said corresponding position first-layer neuron output and an inhibitory input in communication with said corresponding position first-parallel-layer neuron output;

generating a second-layer neuron output pulse only when there is a pulse on said excitatory input and an absence of a pulse on said inhibitory input;

establishing n X m third-layer neurons arranged in a matrix of n columns and m rows (n,m), each of said third-layer (k,h) neurons having an excitatory input connected to said corresponding position k second-layer neuron output and an inhibitory input connected to said corresponding h position second-layer neuron output for k = 1 to n and h = 1 to m, where n > m; and

generating a third-layer neuron output pulse at matrix location (k,h) only when there is a pulse on said k column third-layer excitatory input and an absence of a pulse on said h row third-layer inhibitory input for k = 1 to n and h = 1 to m, where n > m.

ABSTRACT OF THE DISCLOSURE

A multi-layered neural network is disclosed that converts an incoming temporally coded spike train into a spatially distributed topographical map from which interspike-interval and bandwidth information may be extracted. This neural network may be used to decode multiplexed pulse-coded signals embedded serially in an incoming spike train into parallel distributed topographically mapped channels. A signal processing and code conversion algorithm not requiring learning is provided.

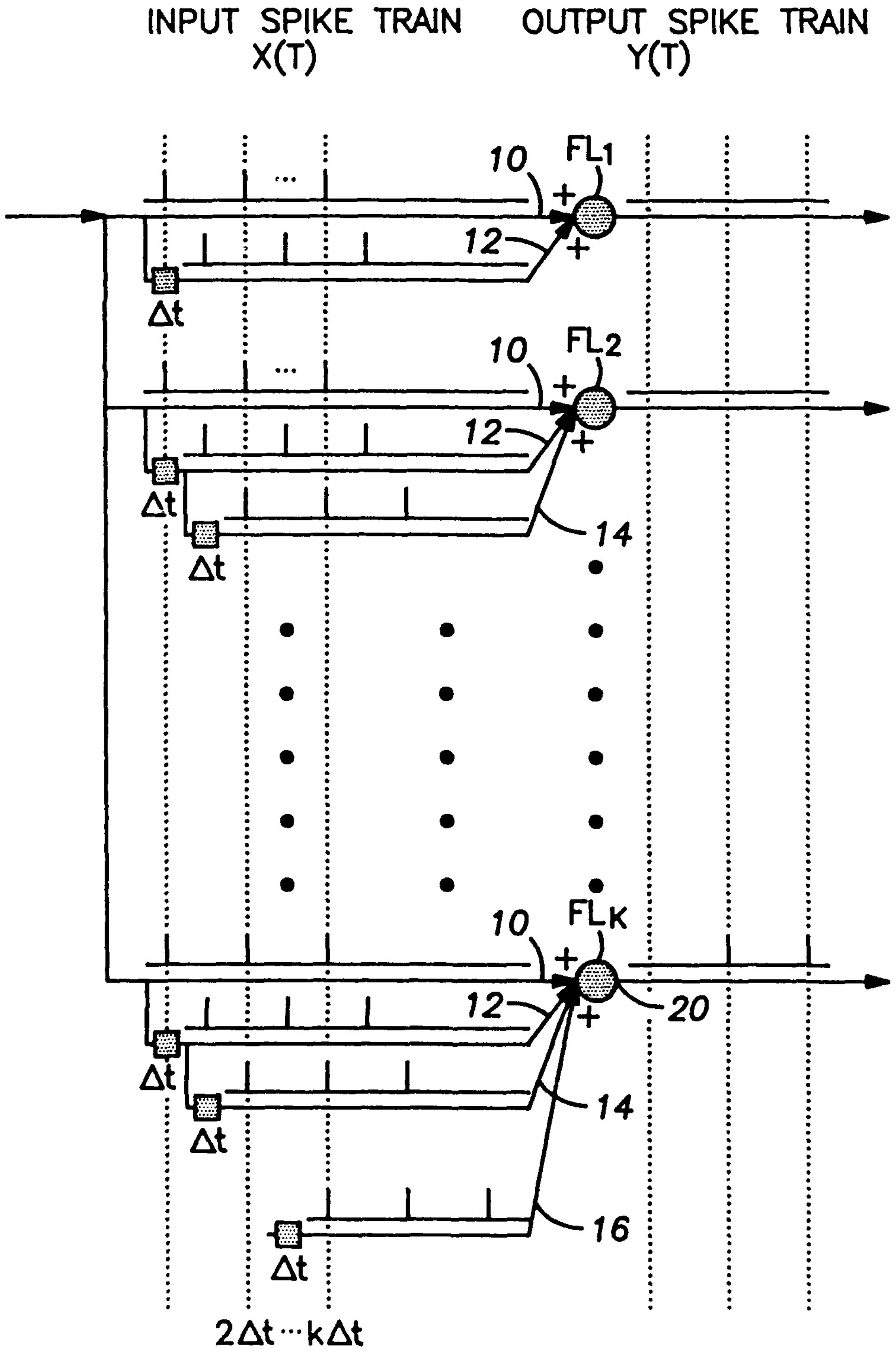

INPUT SPIKE TRAIN
X(T)
OUTPUT SPIKE TRAIN
Y(T)
10
FL1
+
12
+
Δt
...
10
FL2
+
12
+
Δt
Δt
14
10
FLK
+
12
+
20
Δt
Δt
14
16
Δt
2Δt ··· kΔt

FIG. 1

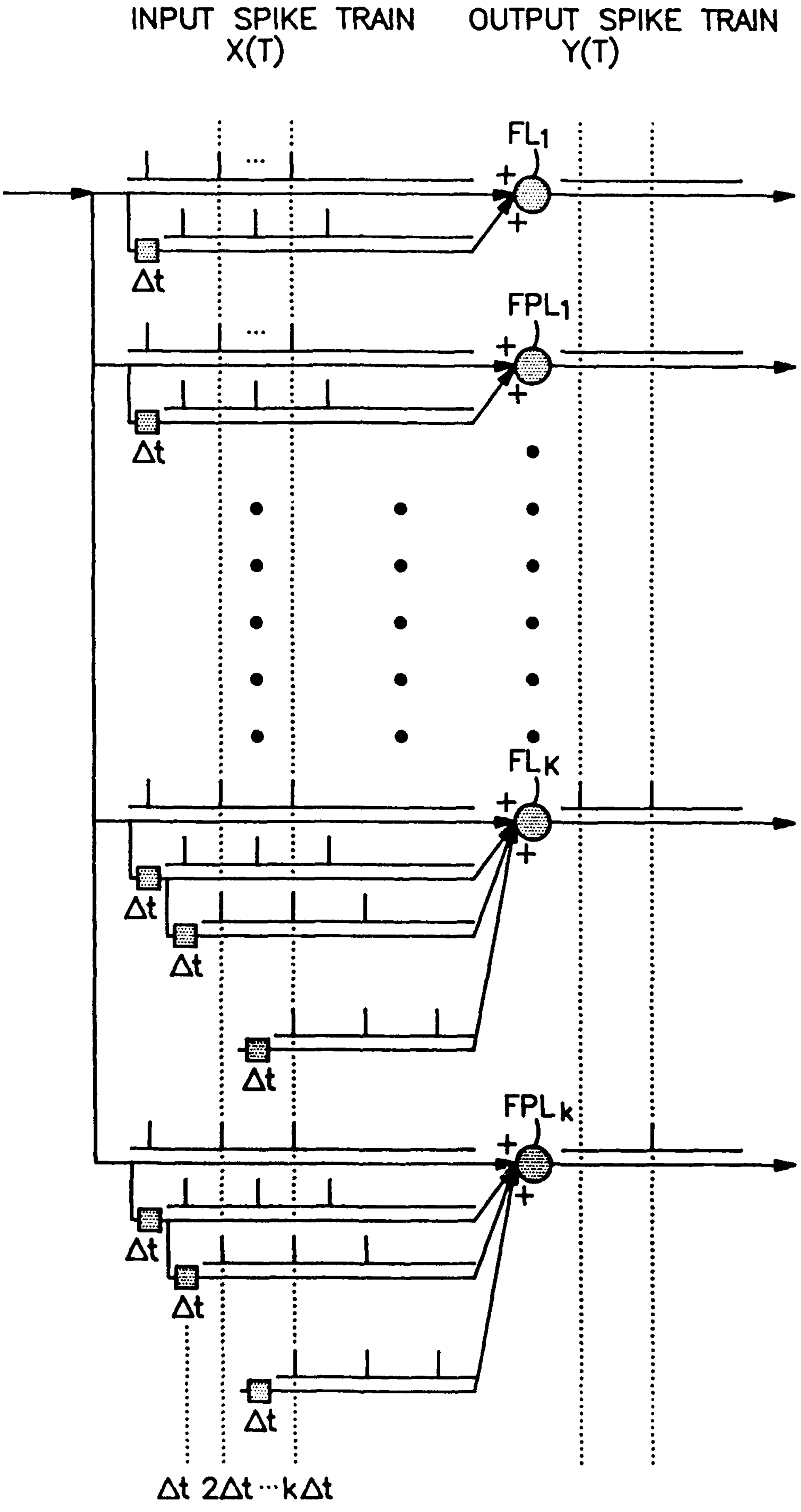

INPUT SPIKE TRAIN
X(T)
OUTPUT SPIKE TRAIN
Y(T)
FL1
+
+
Δt
FPL1
+
+
Δt
FLK
+
+
Δt
Δt
Δt
FPLk
+
+
Δt
Δt
Δt
Δt 2Δt ··· kΔt

FIG. 2

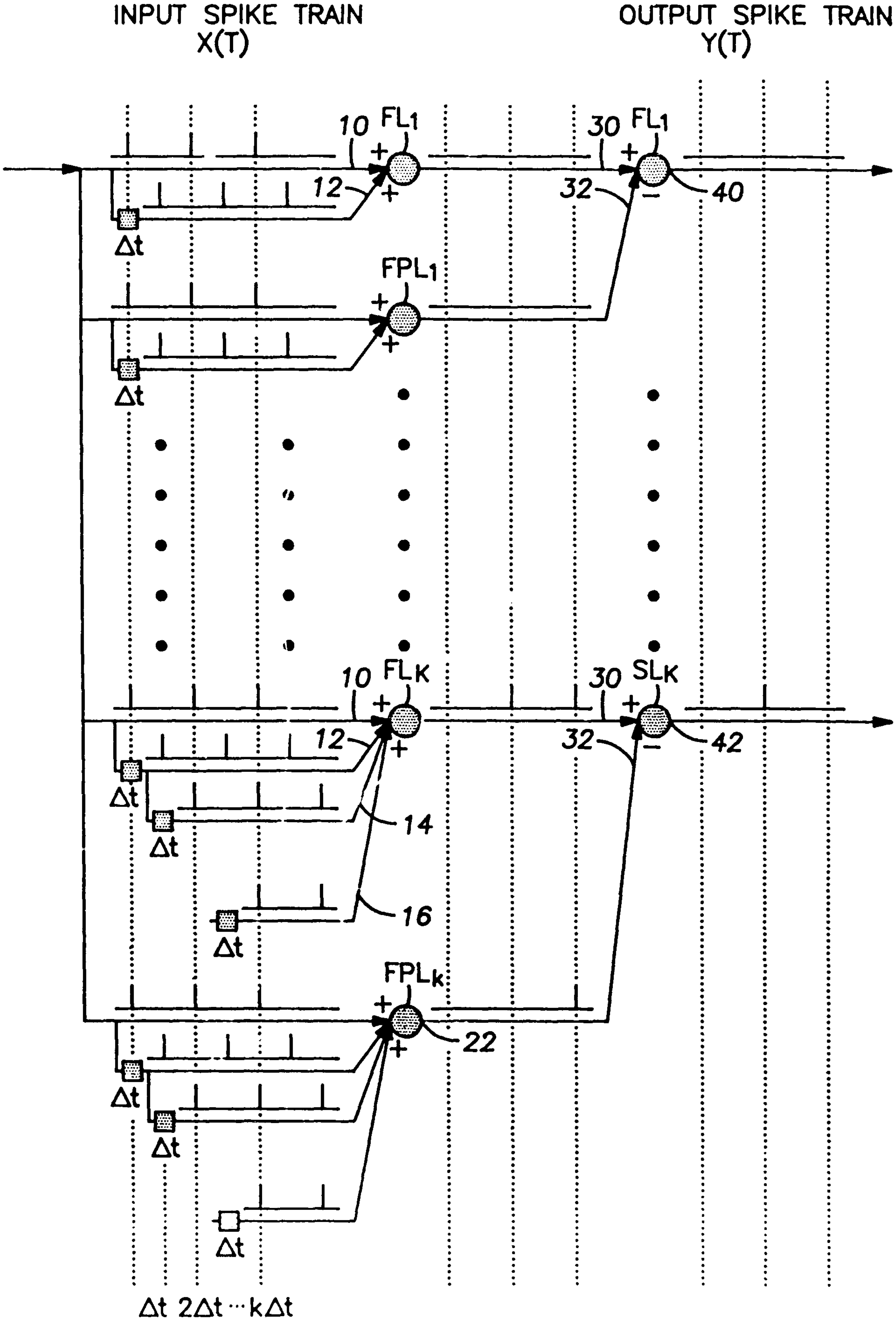

INPUT SPIKE TRAIN
X(T)
OUTPUT SPIKE TRAIN
Y(T)
10
FL1
+
12
+
Δt
30
FL1
+
32
−
40
FPL1
+
+
Δt
10
FLK
+
12
+
Δt
14
Δt
16
Δt
30
SLK
+
32
−
42
FPLk
+
22
+
Δt
Δt
Δt
Δt 2Δt ··· kΔt

FIG. 3

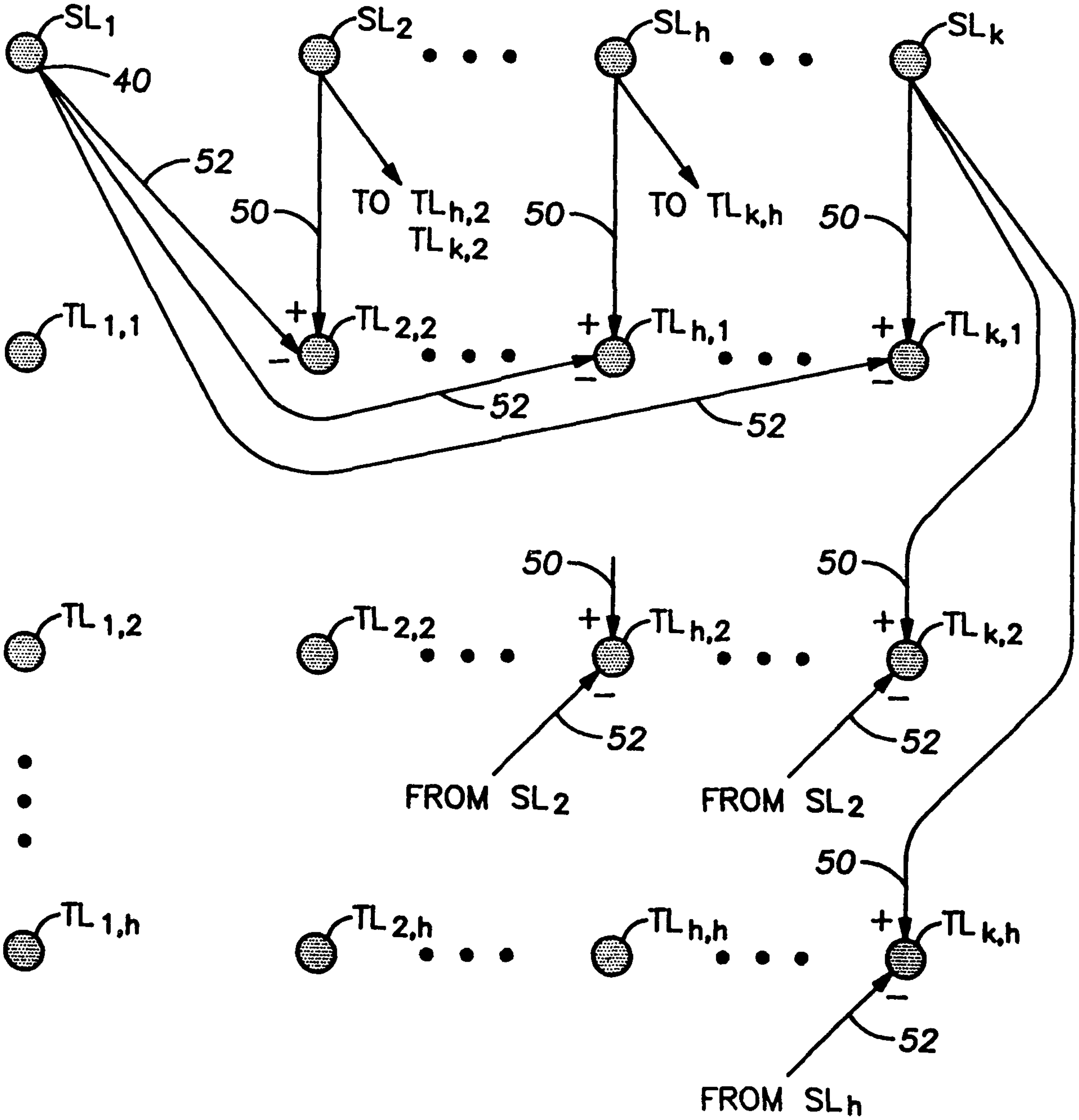
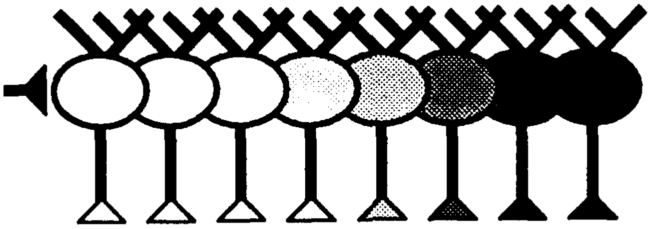

SL$_1$
40
SL$_2$
SL$_h$
SL$_k$
52
50
TO TL$_{h,2}$
TL$_{k,2}$
50
TO TL$_{k,h}$
50
+
TL$_{1,1}$
TL$_{2,2}$
+
TL$_{h,1}$
+
TL$_{k,1}$
−
−
−
52
52
50
50
+
TL$_{1,2}$
TL$_{2,2}$
TL$_{h,2}$
+
TL$_{k,2}$
−
−
52
52
FROM SL$_2$
FROM SL$_2$
50
TL$_{1,h}$
TL$_{2,h}$
TL$_{h,h}$
+
TL$_{k,h}$
−
52
FROM SL$_h$

FIG. 4

MacNeuron

Tutorial



0 The MacNeuron Tutorial

0. 1 What is MacNeuron?

MacNeuron is a tool for simulating custom designed neural systems.

0. 2 About this Tutorial

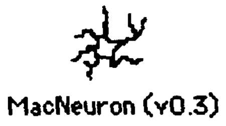
This tutorial provides a self contained introduction to the *MacNeuron* application. Each example utilizes a step by step approach, allowing you to follow along on your own Macintosh computer. After completing this tutorial, you should feel confident enough to begin experimenting with *MacNeuron* on your own. Also consult the *MacNeuron Reference Manual* for a comprehensive review of all the features built in to the *MacNeuron* application.

note: *MacNeuron* is still very much in development and many features that will ultimately be part of the application have not yet been incorporated or are still getting the 'bugs' worked out . To avoid rewriting this tutorial with each revised version of *MacNeuron*, this tutorial has been written, as much as possible, with the 'final' version in mind. Occasionally, this creates discrepancies between what is written in the tutorial and what is actually appearing on the screen. Where such discrepancies occur, temporary text, such as this, may be inserted to clear up possible confusion. In such cases, the tutorial should be read as a guide to the final form that *MacNeuron* is expected to take.

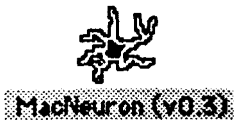
1 Example I: Running the *MacNeuron* Application

1. 1 Getting Started

The *MacNeuron* icon (version 0.3) looks like this:



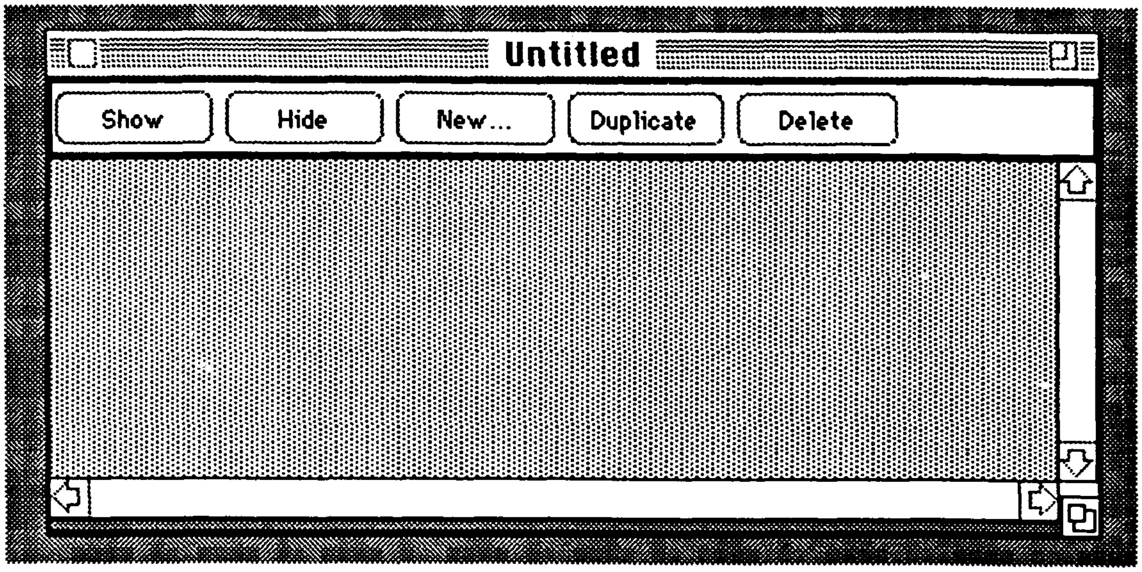
Click on the *MacNeuron* icon to select it:



Start *MacNeuron* by double clicking on the *MacNeuron* icon. (Alternatively, you can select the *MacNeuron* icon and then choose "*Open...*" from the *File* menu).

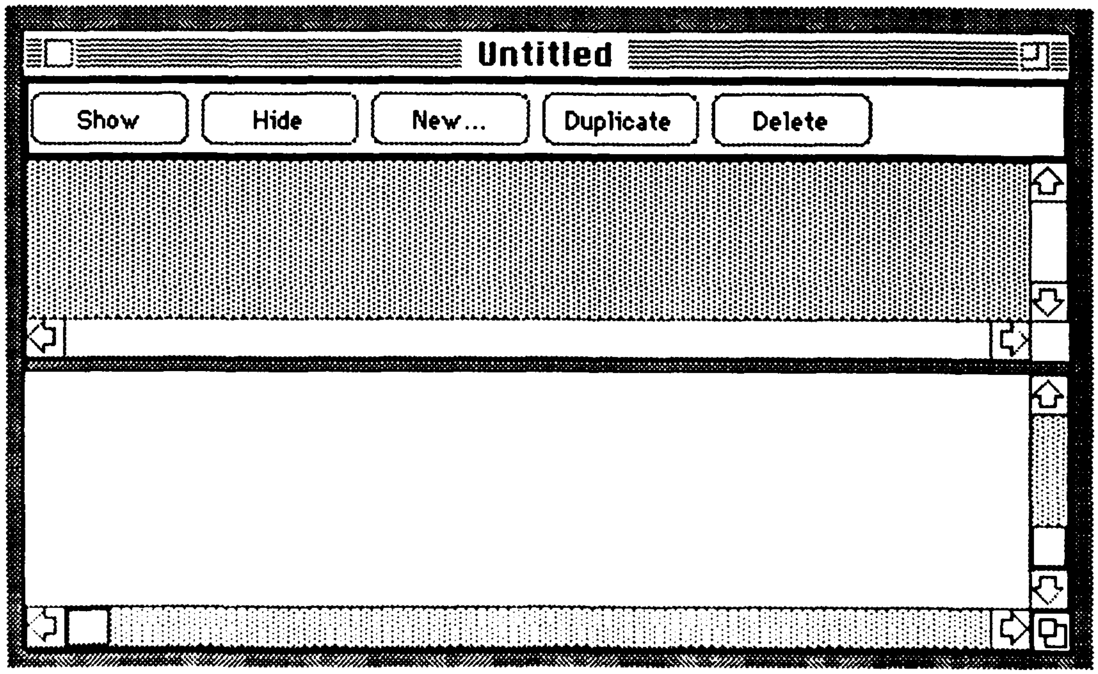
1.2 Main *MacNeuron* Window

MacNeuron will start up with a *New* window called *Untitled*. This is the *main window* of the application. The name of this main window will change when the file is saved, just like any other standard Macintosh program.



1.3 Resizing the Main *MacNeuron* Window Partitions

This main window contains two parts (or partitions). When the resize box at the lower right corner of the window is enlarged, the second half (lower partition) of the window will show up:



By holding down the mouse button at the window-partition divider (the shaded thick line between the two partitions) and "draging" the divider up or down, the relative partition sizes can be adjusted. Note that the cursor changes from a pointer to a cross-hair when it is on the window partition divider.

1.4 Simulation List (Upper Partition)

The upper partition of this main window consists of nested lists containing all the objects (neurons, conductances, compartments, networks, drivers, graphs, etc...) currently in the simulation. The upper portion of the window also provides a palette of tools for performing useful operations on the listed items. For instance, selecting a particular neuron from the list and then clicking the show button will cause the window for that neuron to be displayed. There are also tools for hiding, deleting and duplicating any of the objects in the list. Examples of these operations are provided later in the tutorial.

1.5 Simulation Language (Lower Partition)

The lower partition of the main window is a text window where circuits may be specified using a high level language, called "*macro*", which provides the same functionality as the iconic user interface, but offers a powerful alternative for performing large numbers of similar operations.

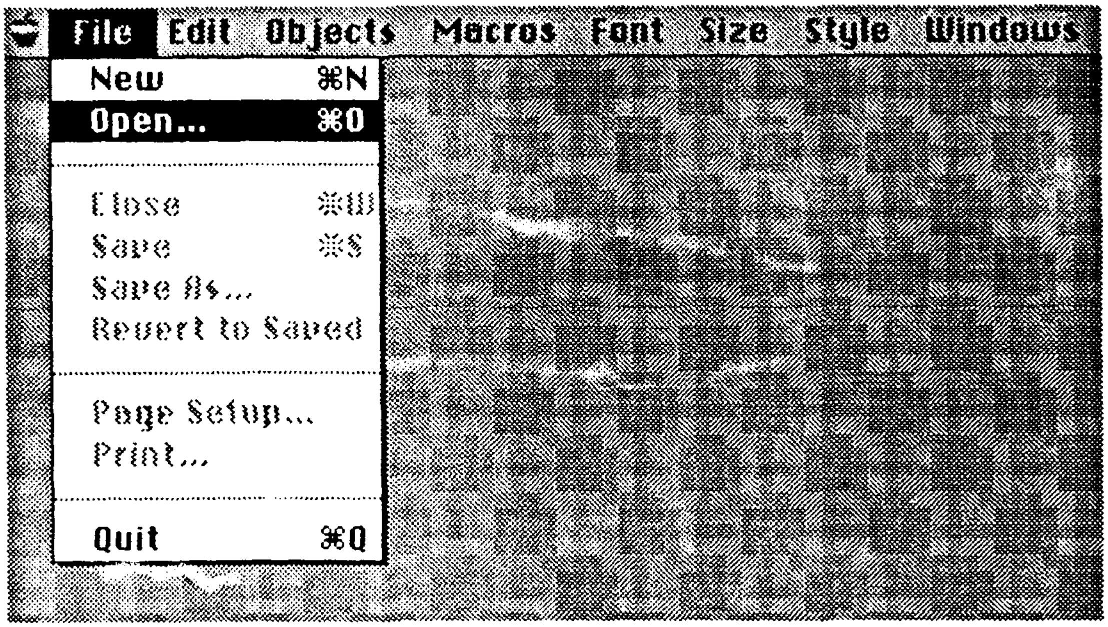
1.6 What ‘s Next?

We are now ready to go the next example. To close the *Untitled* window, click in the close box in the upper left corner. It is not necessary to close the *Untitled* window in order to proceed, but doing so will reduce unnecessary clutter on your screen.

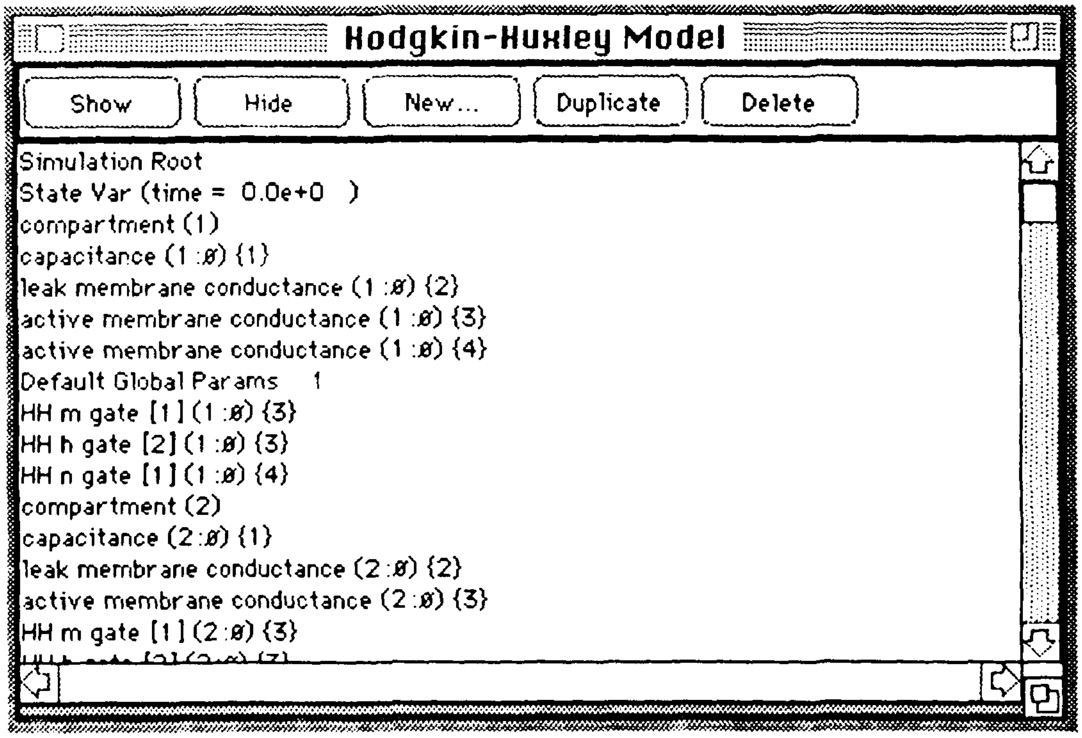
2 Example II: Simulating a Previously Constructed Neural Circuit

2.1 Opening Existing Files

At the top of the screen is the menu bar. To open an existing file, pull down the *File* menu from the menu bar and select “*Open...*”



A standard dialog box will appear asking you to select a file. Open the file called *Hodgkin-Huxley Model* (it would be a good idea copy this file first) :



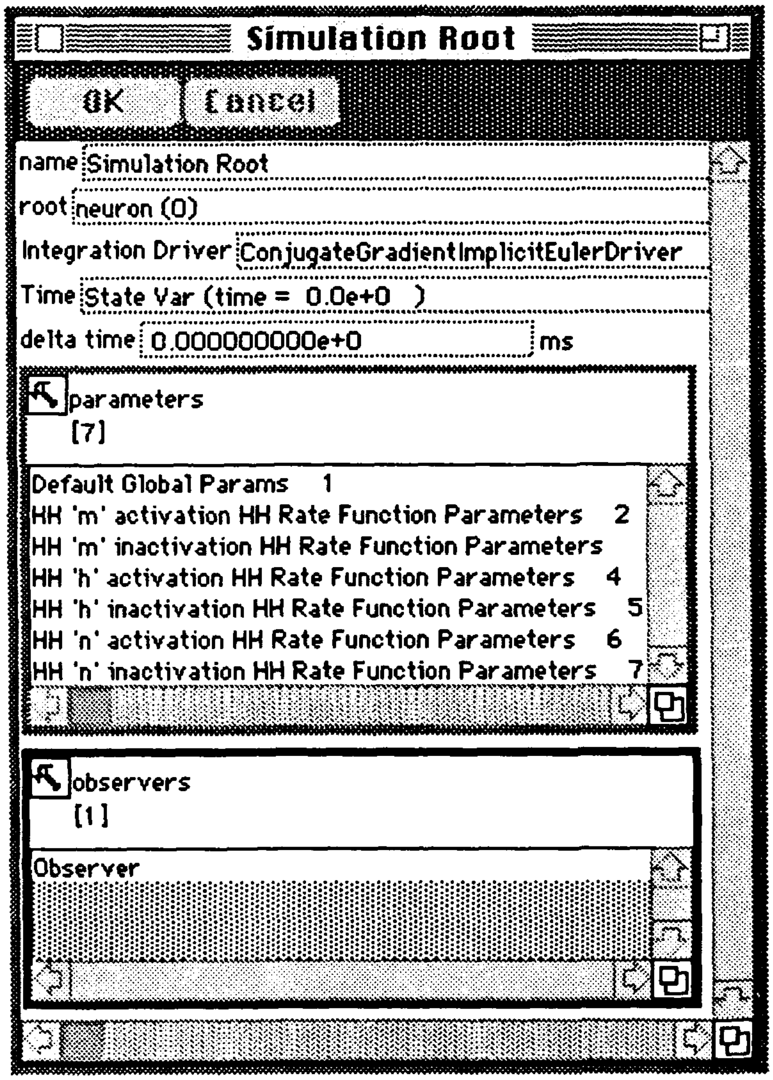
```
Simulation Root
State Var (time =  0.0e+0  )
compartment (1)
capacitance (1 :ø) {1}
leak membrane conductance (1 :ø) {2}
active membrane conductance (1 :ø) {3}
active membrane conductance (1 :ø) {4}
Default Global Params    1
HH m gate [1] (1 :ø) {3}
HH h gate [2] (1 :ø) {3}
HH n gate [1] (1 :ø) {4}
compartment (2)
capacitance (2 :ø) {1}
leak membrane conductance (2 :ø) {2}
active membrane conductance (2 :ø) {3}
HH m gate [1] (2 :ø) {3}
HH h gate [2] (2 :ø) {3}
```

Hodgkin-Huxley Model is a two compartment circuit representing a short axonal segment containing active membrane conductances. The main window contains a single object called “Smulation Root’”. As we will see, everything in the simulation “grows out” of the Simulation Root, which therefore acts as a landmark from which any other object in the simulation can be found.

The simulation list currently contains all the objects in the simulation. Usually, the *Simulation Root* is the first object in the list.

2.2 *Simulation Root* Window

Double click on the *Simulation Root* to open its window (alternatively, select the *Simulation Root* by clicking on it once and then click "*Show*" in the main window's command pane):



name Simulation Root
root neuron (0)
Integration Driver ConjugateGradientImplicitEulerDriver
Time State Var (time = 0.0e+0)
delta time 0.000000000e+0 ms

parameters
[7]

Default Global Params 1
HH 'm' activation HH Rate Function Parameters 2
HH 'm' inactivation HH Rate Function Parameters
HH 'h' activation HH Rate Function Parameters 4
HH 'h' inactivation HH Rate Function Parameters 5
HH 'n' activation HH Rate Function Parameters 6
HH 'n' inactivation HH Rate Function Parameters 7

observers
[1]

Observer

You will have to manipulate the various panes inside the *simulation root* window in order to achieve the configuration shown. Currently, the default window organization is not ideal. This will be the case with essentially all the windows appearing in the tutorial.

The *simulation root* points to all the main object categories, or hierarchies, which make up the simulation:

root neuron (0) : The top of the physiological hierarchy. This could also be a network or a brain.

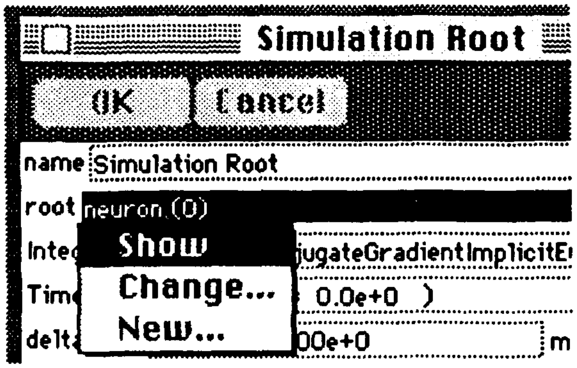
Integration Driver ConjugateGradientImplicitEulerDriver : The object which implements the particular numerical integration algorithm chosen to drive the simulation. The currently selected driver uses conjugate gradient descent to solve a matrix of coupled linear equations resulting from an implicit integration time step.

parameters : Objects in this list contain parameters which can be shared by multiple simulation objects. This allows the same parameters, such as those characterizing voltage dependent conductance channels, to be used by more than one voltage-activated gate.

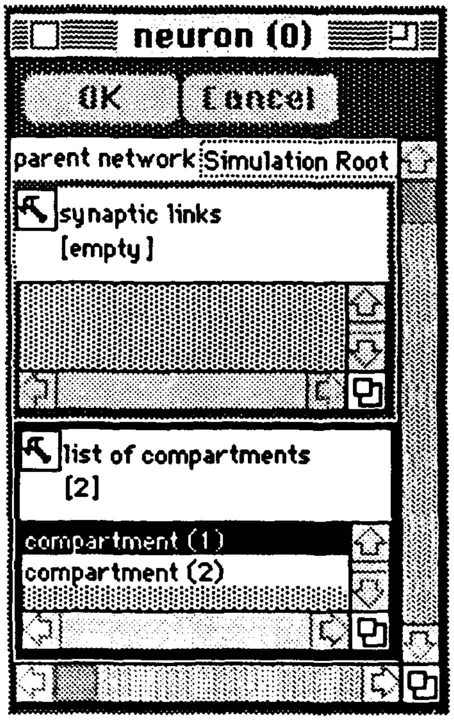
observers : List of objects which contain other objects for recording and ploting simulation data.

2.2 *Root Neuron* Window

Select *Show* from the root neuron's pull down menu to activate its window:



name Simulation Root
root neuron (0)
Inte jugateGradientImplicitE
Tim 0.0e+0)
delt 00e+0 m

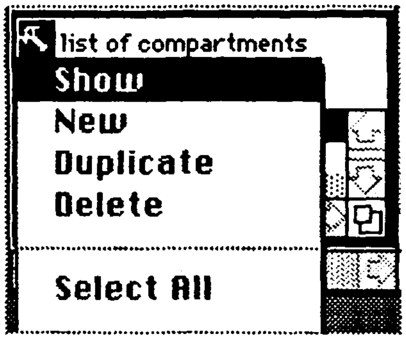


parent network Simulation Root
synaptic links
[empty]
list of compartments
[2]
compartment (1)
compartment (2)

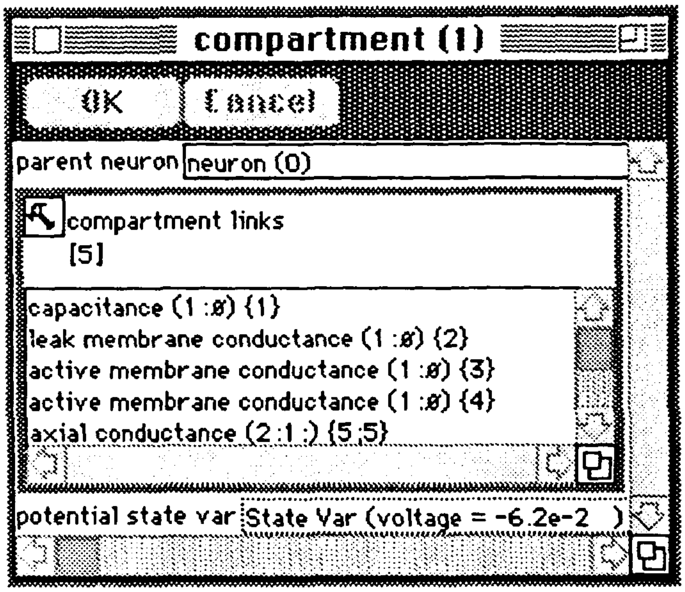
Neuron (0) contains two compartments and has no synaptic links. Since *neuron (0)* is the root physiological object, the network which contains it (parent network) is the *simulation root.* Click on the text "*compartment (1)*" (in the list of compartments sub-window) to highlight it:

2.2 Compartment Window

Activate the window for *compartment (1)* by double clicking on its text or by selecting Show from the tools pull down menu (You may also wish to close the *neuron (0)* window as it will no longer be needed).



Compartment (1) displays its parent neuron, in this case the *root neuron, neuron (0),* a list of links to other compartments (or to "ground" as indicated by a 'ø' symbol) and the electrical potential. Keep this window available since we will observe the potential in this compartment during the simulation:



parent neuron neuron (0)

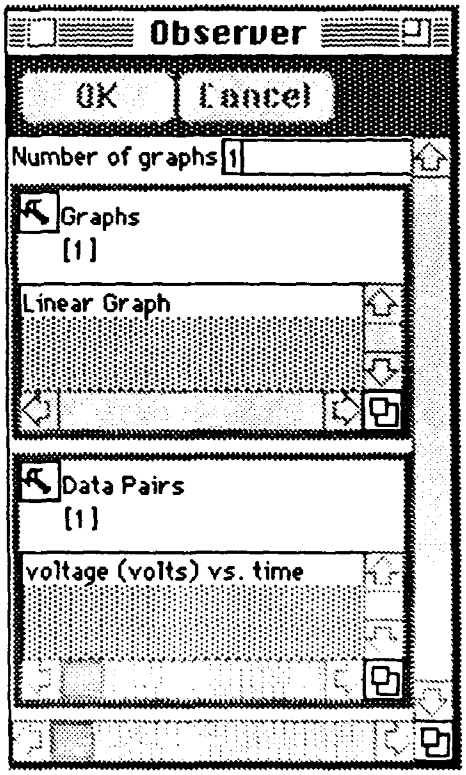
compartment links
[5]

capacitance (1 :ø) {1}
leak membrane conductance (1 :ø) {2}
active membrane conductance (1 :ø) {3}
active membrane conductance (1 :ø) {4}
axial conductance (2:1:) {5;5}

potential state var State Var (voltage = -6.2e-2)

2.3 Observers

Return to the *Simulation Root* and activate the *Observer* located in the *Observers* subwindow:



Number of graphs 1

Graphs
[1]

Linear Graph

Data Pairs
[1]

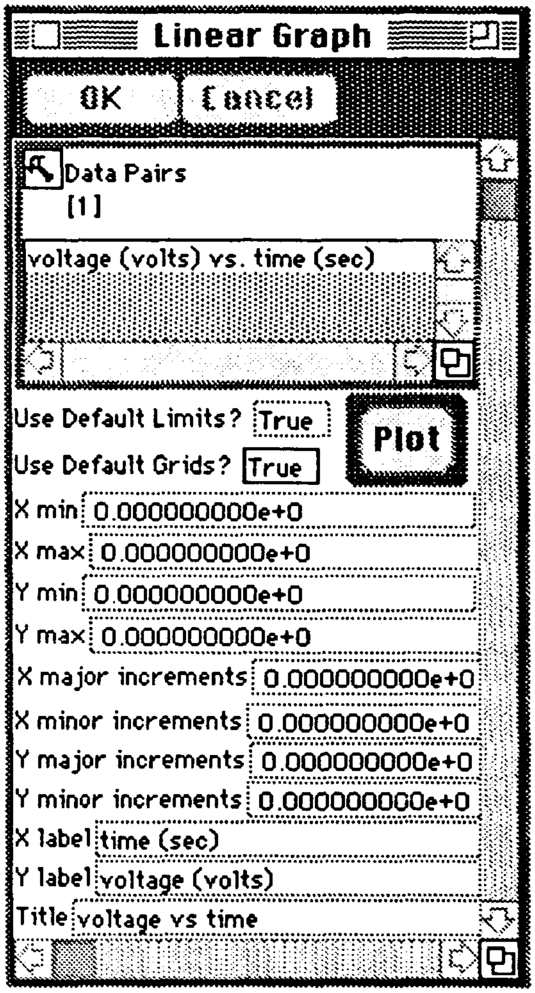
voltage (volts) vs. time

The *Observer* watches the specified *Data Pairs* and ensures that new values are recorded as the simulation progresses. The *Observer* also keeps track of the different graphs employed to display the data.

2.4 Graphs

Many of the graphical capabilities ultimately to be included in *MacNeuron* are still being developed. What is described here represents an intermediate stage. Some of the windows employed represent temporary ‘patches’ and therefore possess minimal functionality. Such windows will be replaced in the next version of *MacNeuron*.

Activate *Linear Graph* found in the *Graph* sub-window (You may also want to close the *Observer* window as it will not be needed for the following).



Data Pairs
[1]
voltage (volts) vs. time (sec)

Use Default Limits? True
Use Default Grids? True
X min 0.000000000e+0
X max 0.000000000e+0
Y min 0.000000000e+0
Y max 0.000000000e+0
X major increments 0.000000000e+0
X minor increments 0.000000000e+0
Y major increments 0.000000000e+0
Y minor increments 0.000000000e+0
X label time (sec)
Y label voltage (volts)
Title voltage vs time

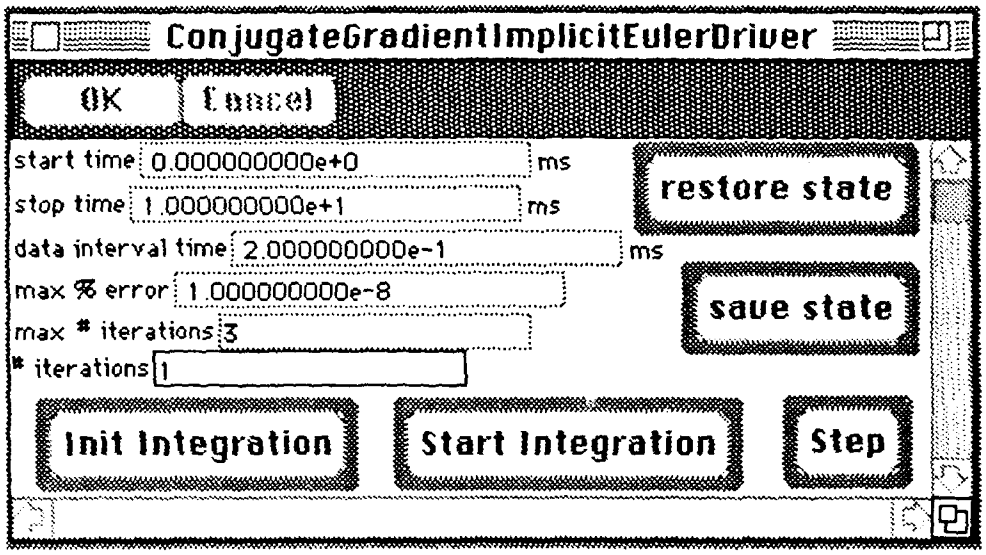
Linear Graph contains a list of those *Data Pairs* to be ploted, as well parameters for specifying the range of data to be included, intervals between major and minor tic marks, axis labels, etc... There is also an option for generating reasonable default values for these parameters.

Save this window as it will be needed to plot the results of the simulation.

2.5 Drivers

Return to the simulation root and activate the *Conjugate Gradient Implicit Euler Driver* located in the *integration driver* sub-window:

The *integration driver* contains sub-windows for specifying the end points of the integration, the length of each integration time step, the maximum percentage error (needs to be set very low due to the inherent stiffness of the circuit), the maximum number of iterations (number of conjugate-gradient descents) per time step and the actual number of iterations used during the previous time step.



start time 0.000000000e+0 ms

stop time 1.000000000e+1 ms

data interval time 2.000000000e-1 ms

max % error 1.000000000e-8

max # iterations 3

iterations 1

There are also several command buttons. A particular state, such as the equilibrium state, can be saved and then later restored using the buttons *"save state"* and *"restore state"*. The *"init integration"* button initializes the integration (used only after everything in the simulation is in place) and the simulation may be performed either in step mode or all at once, using the *"step"* and *"start integration"* buttons, respectively.

2.6 Running

To establish a baseline, hit the *"init integration"* and *"start integration"* buttons in the integration driver window (*Hodgkin-Huxley Model* should have started out in steady state but you can hit the *"restore state"* button in the integration driver window to be sure). The cursor will change to a stop watch while the simulation is running.

At the moment, the cursor does not change shape while the simulation is running and nothing on your screen responds to mouse clicks during this time. Additionally, the integration algorithms have not yet been optimized and therefore integrations may take a little while, depending on which computer you are using (the above baseline calibration takes a few seconds on a IIci).

Activate the widow for *compartment (1)* (use the Window menu if it has become buried).

Alas, the window menu is yet correctly implemented, so you will have to move things out of the way in order to find what you're looking for.

Since we are at steady state, it is necessary to change the potential from the steady state value (in this case by hand) in order to get something going. Select the *potential* sub-window and change the voltage to -40.0 *mV*.

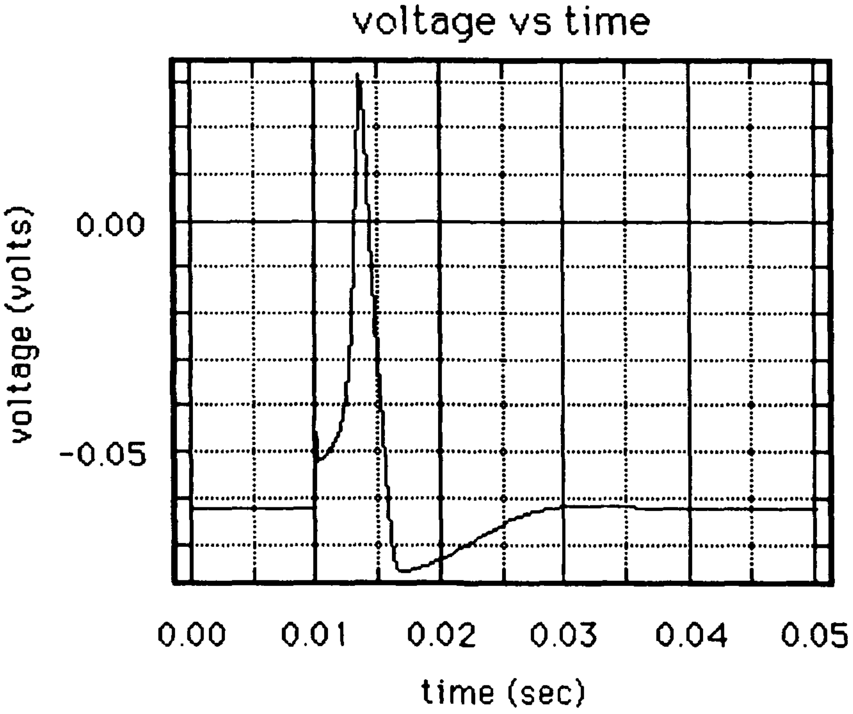
To facilitate development, *State Var* objects have been temporarily introduced into *MacNeuron*. Open a *State Var* just like any other window and edit its value in the usual way. A very important point to remember is that all *State Var* objects use MKS units, so 40.0 *mV* must be written as 0.040. *State Var* objects will be removed in the next version of *MacNeuron*.

You can now close of the *compartment (1)* widow.

Activate the integration driver window, change the start and stop times to 10.0 and 50.0 *msec*, respectively, and restart the integration.

2.7 Ploting

Activate the *Linear Graph* window and hit the *"Plot"* button. A plot of the voltage in *compartment (1)* vs *time* should appear:



Currently, *"Plot"* uses a temporary window with almost no functionality. One pathological feature of this window is that the grow box is hidden. Clicking where you expect the grow bow to be brings it into view. This window does not close in the usual way either. as clicking in the close box does nothing. As mentioned previously, this window is merely a temporary patch while more sophisticated plotting windows are being developed.

2.8 What's Next?

Select *"Close"* from the *File* menu. *MacNeuron* will prompt you to save your changes, including the results of the simulation, and then all windows associated with *Hodgkin-Huxley Model* should disappear.

MacNeuon does not currently save simulation data, so there is no reason to save any of your changes to *Hodgkin-Huxley Model.* This will be fixed in the next version of *MacNeuron.*

Now that we have seen a bit of how *MacNeuron* works, we are ready to learn how to build neural circuits from scratch.

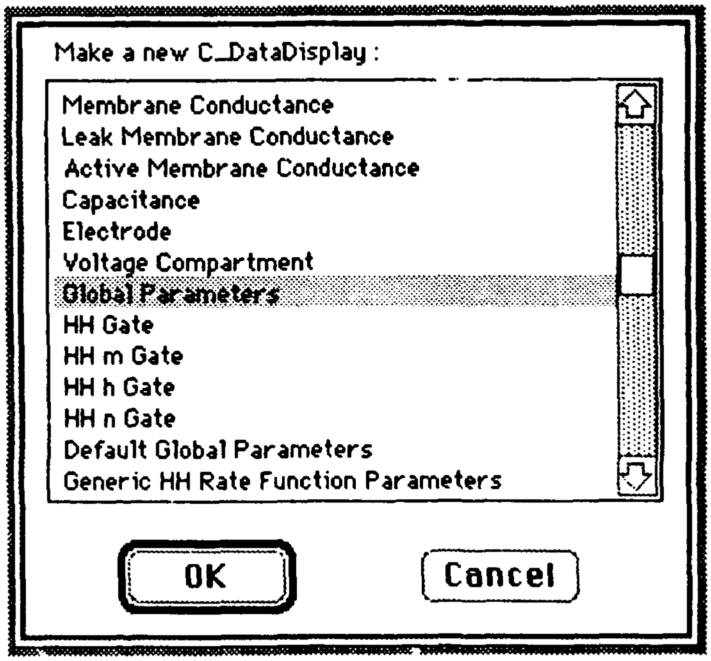
3 Example III: Building Neural Circuits

3.1 New Files

Select *"New..."* from the *File* menu in the *MacNeuron* menu bar at the top of the screen. A new main window, called *"Untitled"*, will appear on the screen (if you have just launched *MacNeuron*, or do so now, the *"Untitled"* window will come up automatically). Choose *Save* from the *File* menu and save this file as Example III.

3.2 Making a *Simulation Root*

The first object in any simulation is the *Simulation Root.* Click on *"New..."* in the main window's command pane. This causes the following dialog box to appear:



Make a new C_DataDisplay :

Only a *Simulation Root* can be made at this point, so this is the only object which appears.

Currently, the user is presented a list containing all possible objects.

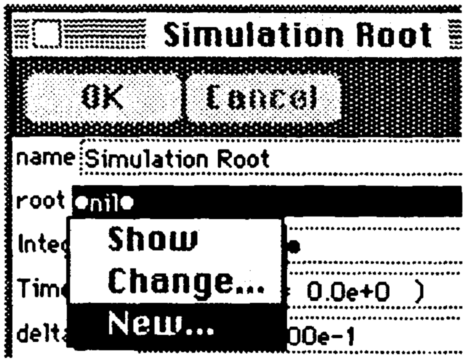
To make a *Simulation Root*, either select *"Simulation Root"* and hit *"OK"* (*"Return"* accomplishes the same thing) or doubie click on *"Simulation Root"*.

Simulation Root may be also called *Global Parameters.*

3.3 Making a *Root Neuron*

Select *"New..."* from the pull down menu in the *Root* sub-widow:

Select *"neuron "* from the *"Make a new"* dialog box and click *"OK"*.

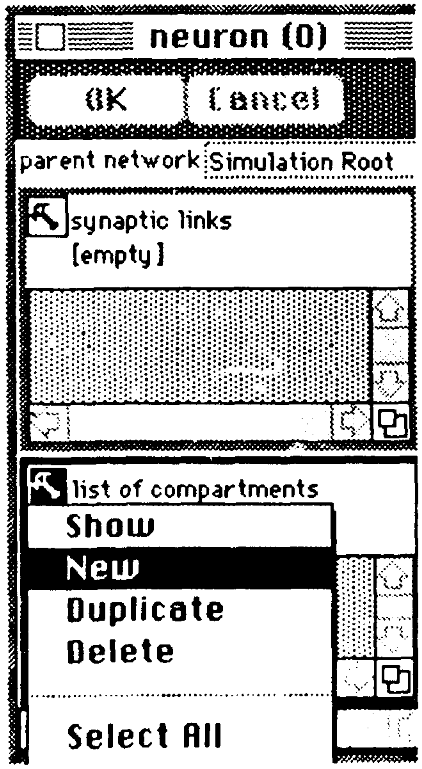


name Simulation Root
root •nil•
Inte •
Tim 0.0e+0)
delt 00e-1

3.4 Making a *Compartment*

Select *"Show"* from the pull down menu in the *Neuron Root* sub-widow containing *neuron (0)* to activate its.window.

Select *"New"* from the *"tools"* pull down menu in the list of compartments sub-widow (in the *neuron (0)*.window):



parent network Simulation Root

Select *"Voltage Compartment"* from the *"Make a new"* dialog box and click *"OK"*.

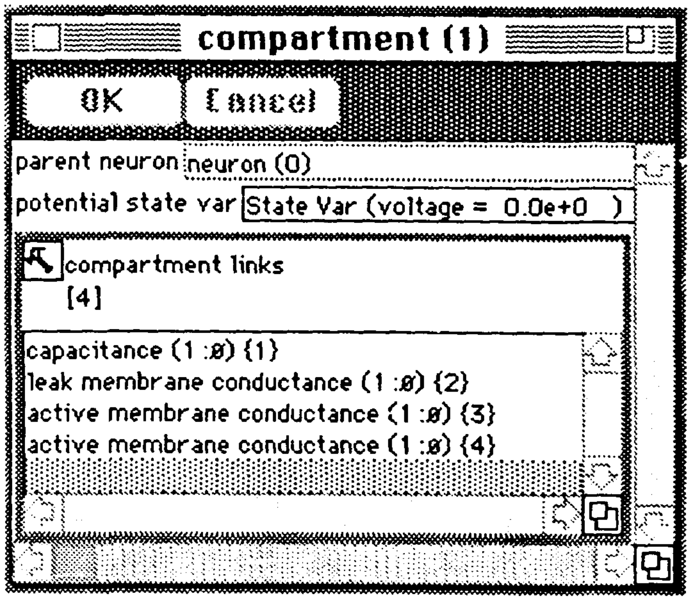
3.5 Making *Compartment Links*

In the *list of compartments* sub-widow, double click on the text *"compartment (1)"* or select *"Show"* from the *"tools"* pull down menu while *"compartment (1)"* is selected.

Select *"New"* from the *"tools"* pull down menu in the compartment links sub-widow (in the *compartment (1)*. window):

Select *"Capacitance"* from the *"Make u new"* dialog box and click *"OK"* .

Make a new *"Leak Membrane Conductance"* and two new *"Active Membrane Conductanc^s"* in the same way. There should now be four items in the *compartment links* sub-widow.



compartment (1)

parent neuron neuron (0)

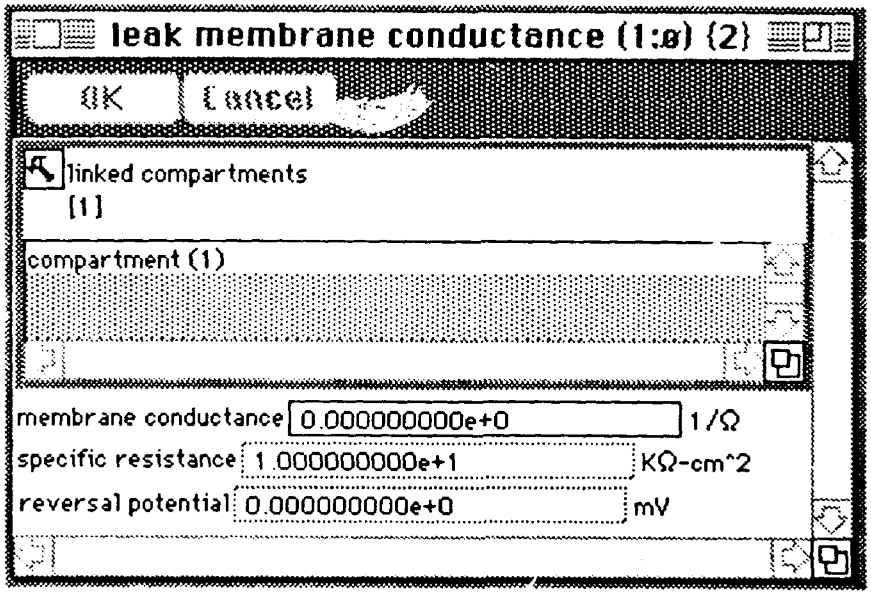
potential state var State Var (voltage = 0.0e+0)

compartment links
[4]

capacitance (1 :ø) {1}
leak membrane conductance (1 :ø) {2}
active membrane conductance (1 :ø) {3}
active membrane conductance (1 :ø) {4}

3.6 Setting the *Leak Membrane Conductance*

Double click on the text *"Leak Membrane Conductance (1:ø) {2}"* in the *compartment links sub-widow* (in the *compartment (1).* window):



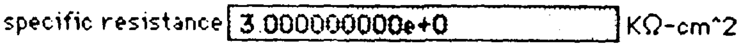
leak membrane conductance (1:ø) {2}

linked compartments
[1]

compartment (1)

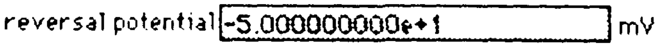
membrane conductance 0.000000000e+0 1/Ω
specific resistance 1.000000000e+1 KΩ-cm^2
reversal potential 0.000000000e+0 mV

Change the *specific resistance* to 3.0 *kΩ-cm^2* and click *"OK"*:



specific resistance 3.000000000e+0 KΩ-cm^2

Change the *reversal potential* to -50.0 *mV* and click *"OK"*:

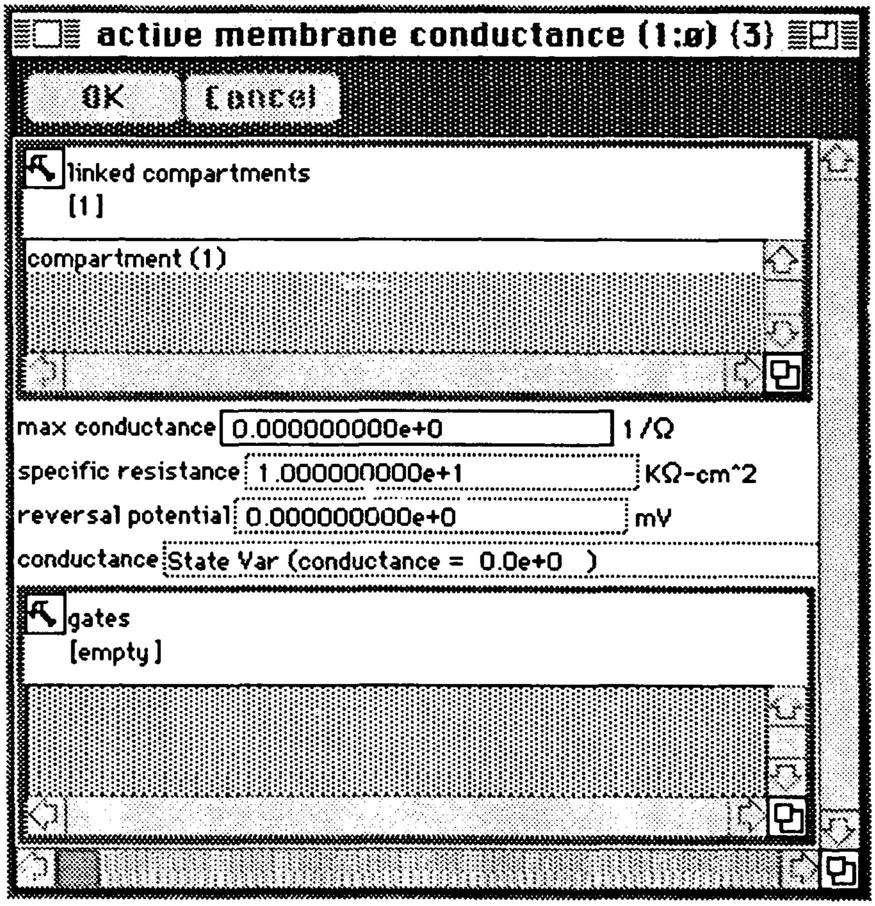


reversal potential -5.000000000e+1 mV

Close the *Leak Membrane Conductance* window.

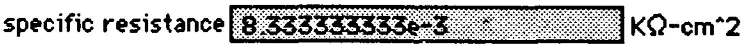
3.7 Setting the *Active Membrane Conductance:* Na$^+$ channel

Double click on the text *"Active Membrane Conductance (1:ø) {3}"* in the *compartment links* sub-widow (in the *compartment (1)*.window):



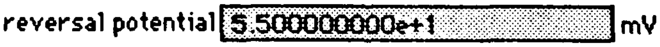
max conductance 0.000000000e+0 1/Ω
specific resistance 1.000000000e+1 KΩ-cm^2
reversal potential 0.000000000e+0 mV
conductance State Var (conductance = 0.0e+0)

Change the *specific resistance* to 0.0083... *kΩ-cm*^2 and click *"OK"*:



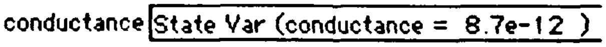
specific resistance 8.333333333e-3 KΩ-cm^2

Change the *reversal potential* to 55.0 *mV* and click *"OK"*:



reversal potential 5.500000000e+1 mV

Change the *conductance* to 8.7e-12 *1/Ω* and click *"OK"*:



conductance State Var (conductance = 8.7e-12)

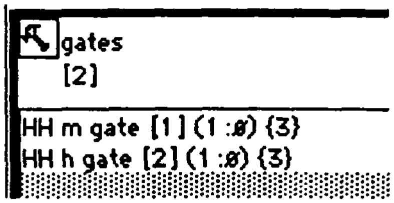
Select *"New"* from the *"tools"* pull down menu in the *gates* sub-widow:

Select *"HH m Gate"* from the *"Make a new"* dialog box and click *"OK"*.

Make a new *"HH h Gate"* as well. There should now be two items in the *gates* sub-widow.

3.8 Setting the *HH m Gate*

Double click on the text *"HH m Gate [1] (1:ø) {3}"* in the gates sub-widow (in the *Active Membrane Conductance (1:ø) {3}* window):



gates
[2]

HH m gate [1] (1:ø) {3}
HH h gate [2] (1:ø) {3}

Gate activation is governed by an equation of the form:

$$dm/dt = a(1-m) - bm$$

where the activation (inactivation) rate is given by α (β). These are also displayed in terms of the steady-state gate activation and the inverse instantaneous time constant.

Select *"New"* from the *"-> rate params"* sub-window:

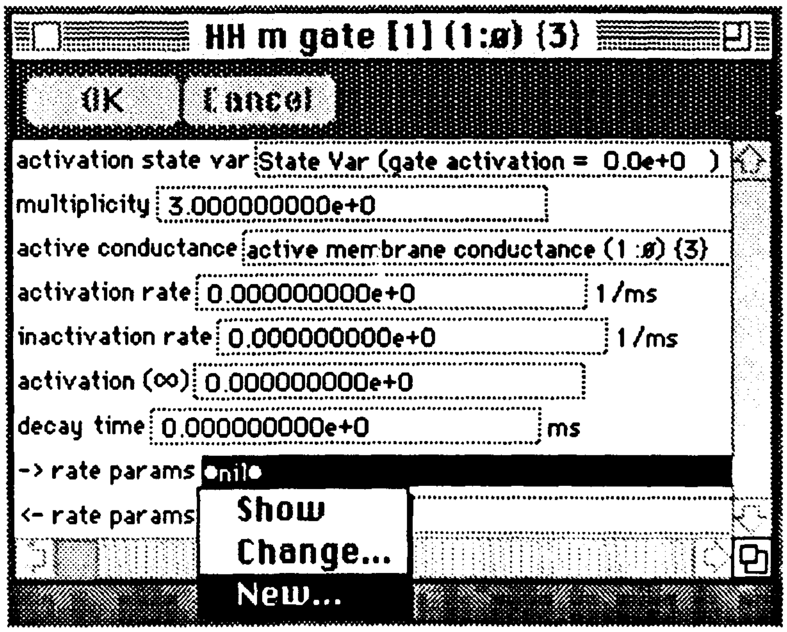
Select *"HH 'm' Gate Activation Rate Function Parameters"* from the *"Make a new"* dialog box and click *"OK"*.

Do the same for the "<- rate params" sub-window, to make a "HH 'm' Gate Inactivation Rate Function Parameters" object.

3.9 Setting the *HH 'm' Gate Activation Rate Function Parameters*

Double click on the text *"HH 'm' Gate Activaiion Rate Function Parameters"* in the *"-> rate params"* sub-widow (in the *HH m Gate [1] (1:ø) {3}* window):

The instantaneous activation rate, α, is governed by an equation of the form:

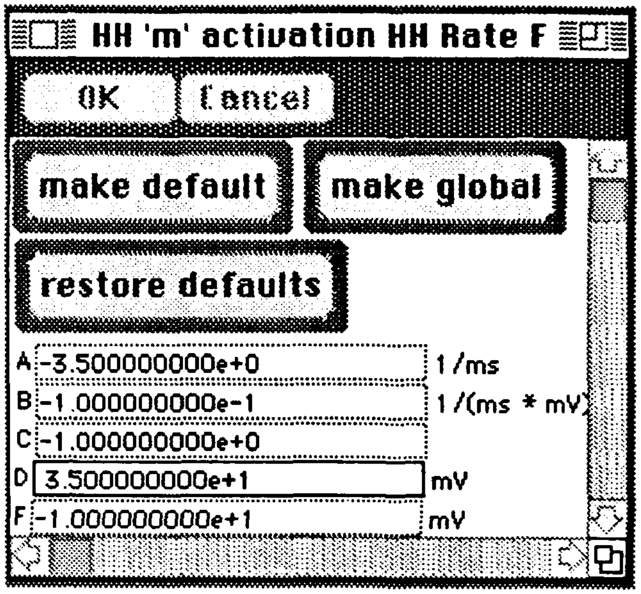


HH m gate [1] (1:ø) {3}

activation state var: State Var (gate activation = 0.0e+0)
multiplicity: 3.000000000e+0
active conductance: active membrane conductance (1:ø) {3}
activation rate: 0.000000000e+0 1/ms
inactivation rate: 0.000000000e+0 1/ms
activation (∞): 0.000000000e+0
decay time: 0.000000000e+0 ms
-> rate params: nil
<- rate params

a(V) = (A + BV)/(C + exp[(V+D)/F]

Set the parameters *A, B, C, D,* and *F* as shown:



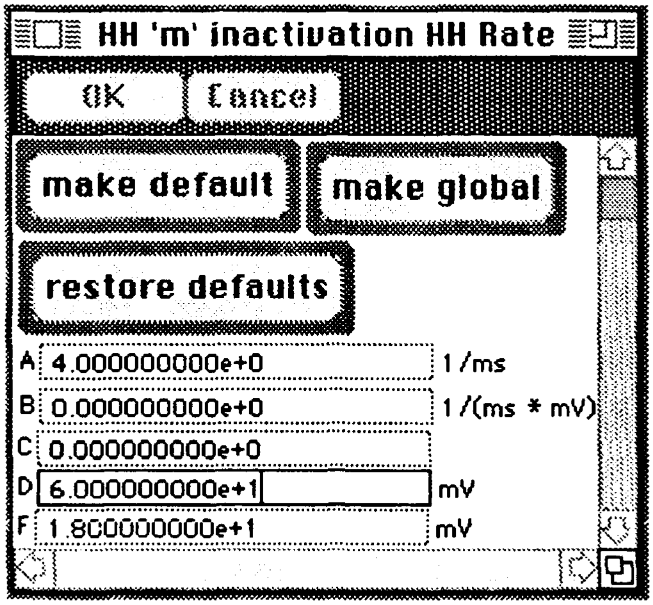
A -3.500000000e+0 1 /ms
B -1.000000000e-1 1 /(ms * mV)
C -1.000000000e+0
D 3.500000000e+1 mV
F -1.000000000e+1 mV

The command button *"make default"* causes this parameter object (and thus these parameters) to be used by all existing *HH m Gates* which do not currently have a parameters object assigned to their *"-> rate params"* sub-widow. The *"make global"* command button makes these parameters universal for all *HH m Gates* regardless of whether or not other parameters have been previously assigned to them. Hitting *"restore defaults"* causes the original parameter values (hard coded into *MacNeuron*) to be restored.

Close the *HH 'm' Gate Activation Rate Function Parameters* window.

3.10 Setting the *HH 'm' Gate Inactivation Rate Function Parameters*

Following 3.9, set the *HH 'm' Gate Inactivation Rate Function Parameters* as shown



A 4.000000000e+0 1 /ms
B 0.000000000e+0 1 /(ms * mV)
C 0.000000000e+0
D 6.000000000e+1 mV
F 1.800000000e+1 mV

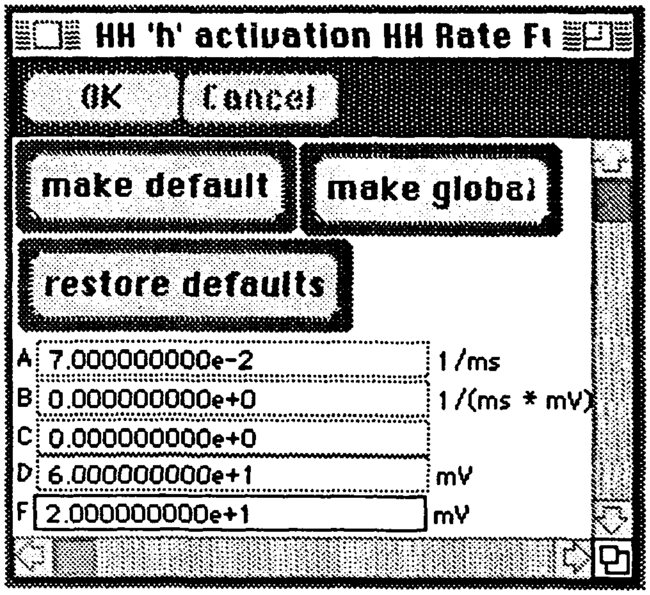
Close the *HH 'm' Gate Inactivation Rate Function Parameters* window.

3.11 Setting the *HH h Gate*

Follow the steps analogous to those in 3.8-3.10 to configure the *HH h Gate [1] (1:ϕ) {3}.*

3.12 Setting the *HH 'h' Gate Activation Rate Function Parameters*

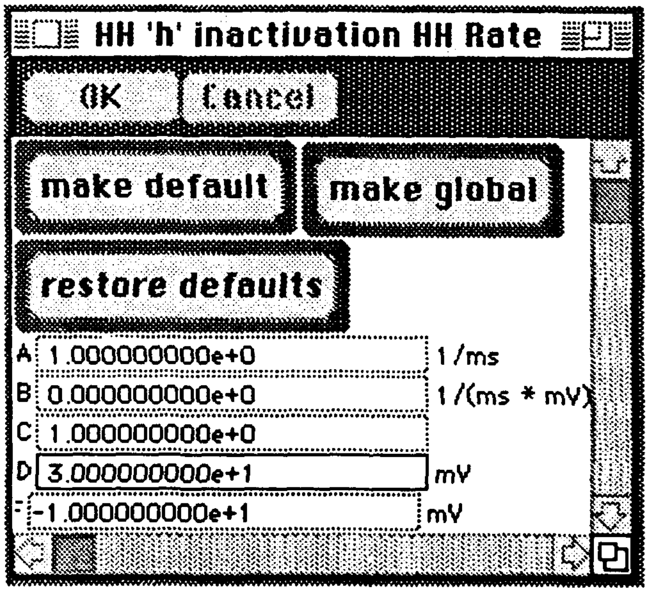
Set the *HH 'h' Gate Activation Rate Function Parameters* as shown:



A 7.000000000e-2 1/ms
B 0.000000000e+0 1/(ms * mV)
C 0.000000000e+0
D 6.000000000e+1 mV
F 2.000000000e+1 mV

3.13 Setting the *HH 'h' Gate Inactivation Rate Function Parameters*

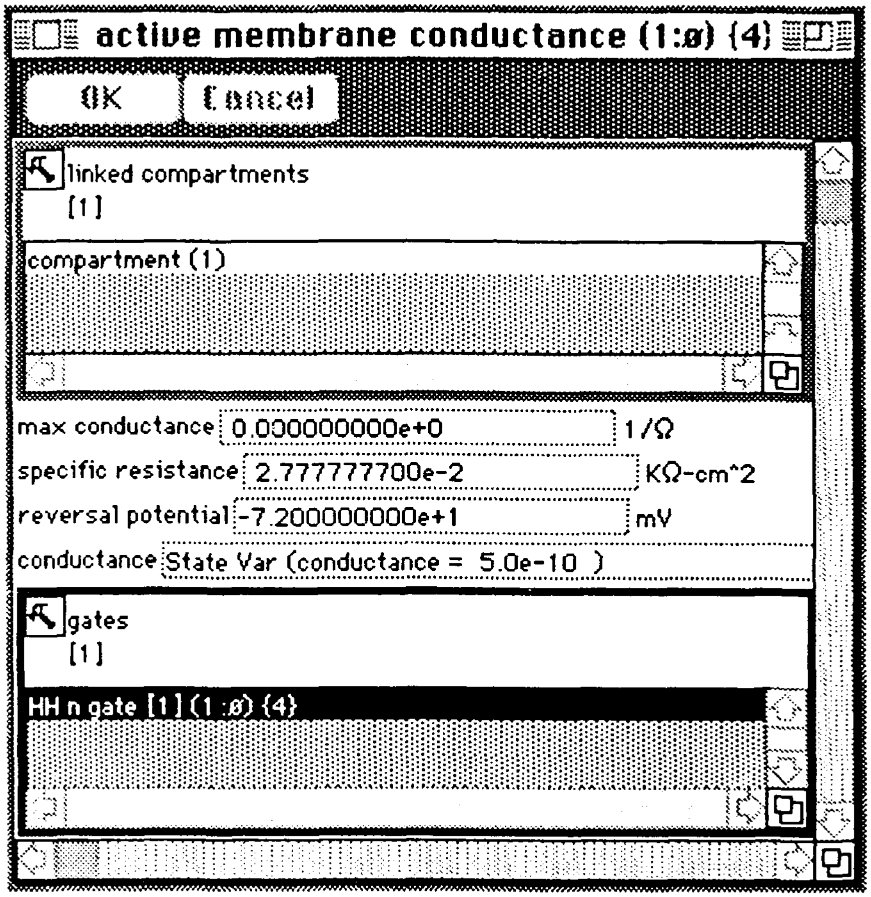
Set the *HH 'h' Gate Activation Rate Function Parameters* as shown:



A 1.000000000e+0 1/ms
B 0.000000000e+0 1/(ms * mV)
C 1.000000000e+0
D 3.000000000e+1 mV
F -1.000000000e+1 mV

3.14 Setting the *Active Membrane Conductance*: K^+ channel

As in 3.7, configure the *"Active Membrane Conductance (1:ø) {4}"* as follows:



active membrane conductance (1:ø) {4}

linked compartments
[1]

compartment (1)

max conductance 0.000000000e+0 1/Ω

specific resistance 2.777777700e-2 KΩ-cm^2

reversal potential -7.200000000e+1 mV

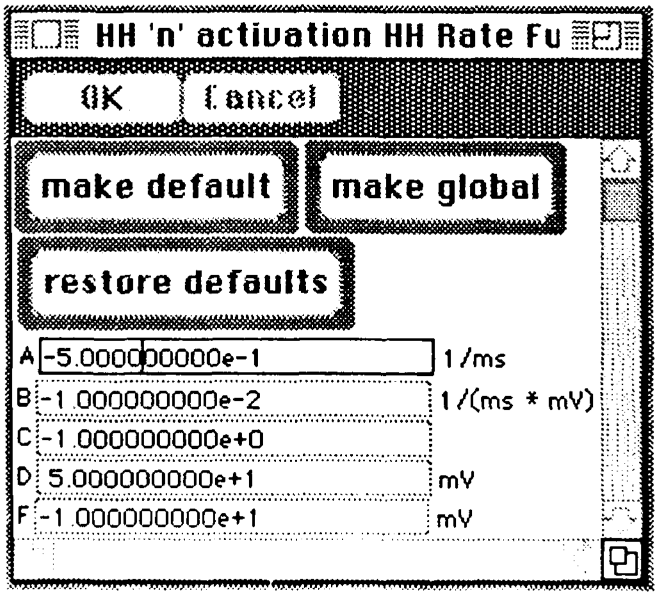
conductance State Var (conductance = 5.0e-10)

gates
[1]

HH n gate [1] (1 :ø) {4}

3.15 Setting the *HH 'n' Gate Activation Rate Function Parameters*

Set the *HH 'n' Gate Activation Rate Function Parameters* as shown:



HH 'n' activation HH Rate Fu

A -5.000000000e-1 1/ms

B -1.000000000e-2 1/(ms * mV)

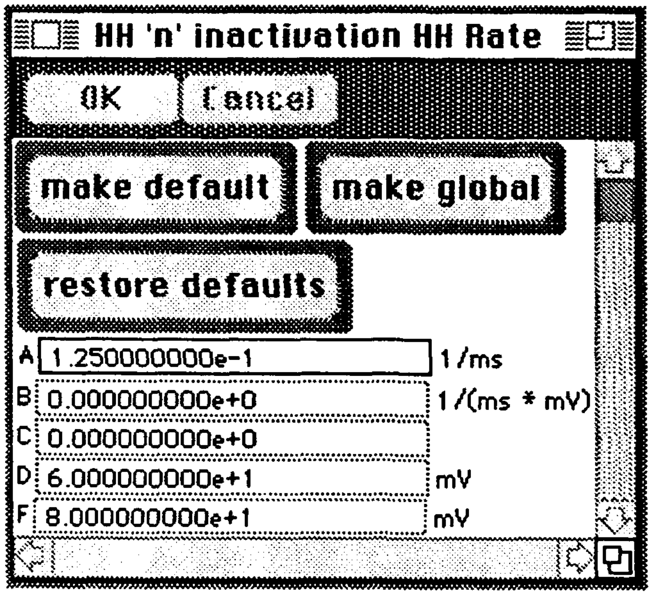
C -1.000000000e+0

D 5.000000000e+1 mV

F -1.000000000e+1 mV

3.16 Setting the *HH 'n' Gate Inactivation Rate Function Parameters*

Set the *HH 'n' Gate Inactivation Rate Function Parameters* as shown:

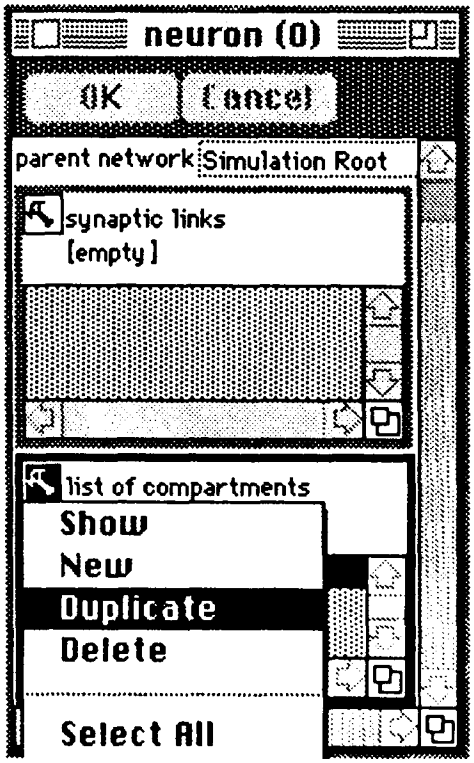


A 1.250000000e-1 1/ms
B 0.000000000e+0 1/(ms * mV)
C 0.000000000e+0
D 6.000000000e+1 mV
F 8.000000000e+1 mV

3.17 Duplicating Compartments

We have now built one complete Hodgkin-Huxley compartment, containing a capacitance, a leak membrane resistance, a voltage-gated Na$^+$ channel, and a voltage-gated K$^+$ channel. To construct a short length of axonal fiber, we string together two such channels.

Does this mean, you ask in despair, that we have to go through the labor of building another compartment? Fortunately, this is not necessary, because *MacNeuron* has sophisticated built in mechanisms for duplicating objects with complex internal structure. To see this, activatethe root neuron window *(neuron (0))* and select the *compartment (1)* object located in the list of compartments sub-window. An identical compartment can be added to the neuron simply by choosing *"duplicate"* from the tools pull down menu:



parent network Simulation Root

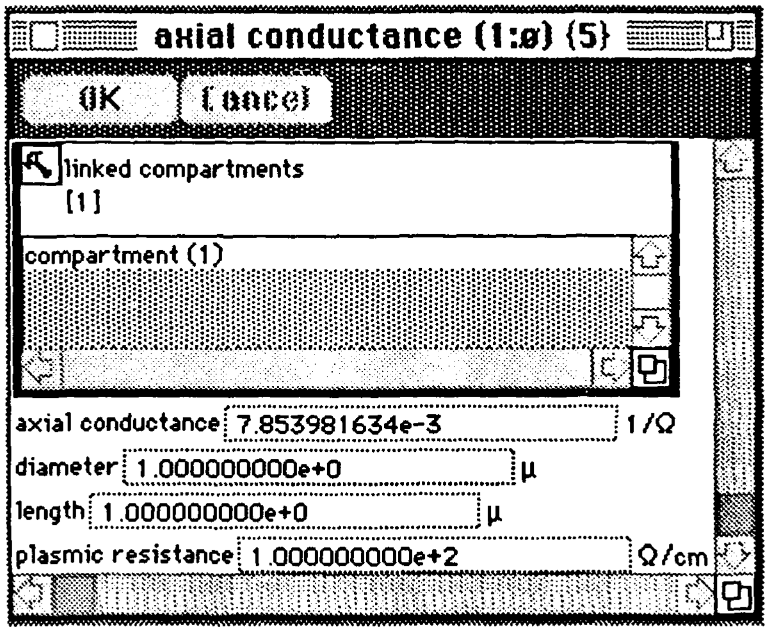
Automatically, a new compartment, identical to the first, is created and added to the root neuron. All of the internal structure of the original is reproduced as well. This is a general feature of *MacNeuron*: Whenever a complex object is duplicated, whether it is a compartment, a neuron, or a network, the entire hierarchy of internal structure is reproduced.

3.18 Chaining Compartments Together with *Axial Conductances*

All we have left to do join the two compartments together.

Activate the window foı *compartment (1)* and select *New* from the *compartment links* pull down menu. When the *Make a new* dialog box appears, choose *Axial Conductance* and click *OK*.

Activate the Axial Conductance window:



axial conductance (1:ø) {5}

linked compartments
[1]

compartment (1)

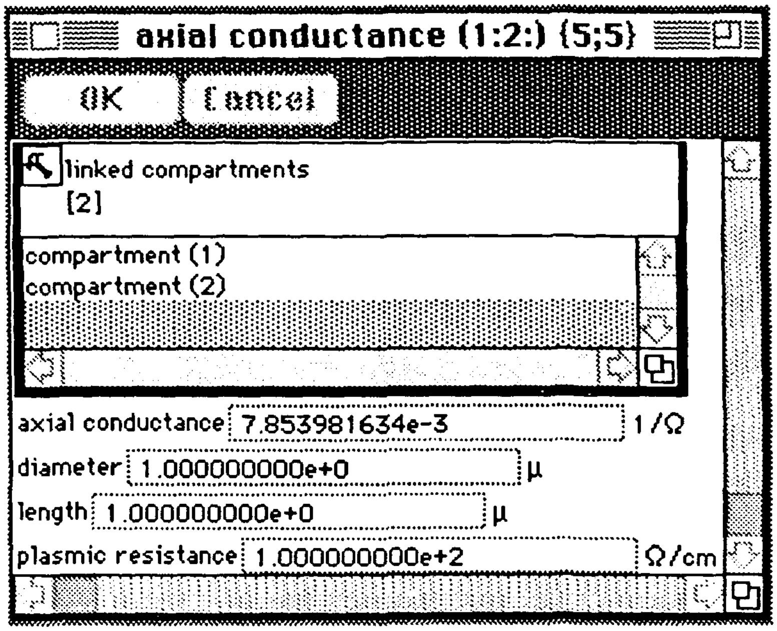
axial conductance 7.853981634e-3 1/Ω
diameter 1.000000000e+0 μ
length 1.000000000e+0 μ
plasmic resistance 1.000000000e+2 Ω/cm

As it stands, the *Axial Conductance* connects *compartment (1)* to ground. In order to join the other end of the axial conductance to the second compartment, we use a drop in technique.

3.19 Moving Objects to and from Other Objects

Parameter values and objects can be moved to and from other objects easily. All it needs is to hold down the option-key while pressing the mouse button over the selected item. An arrow icon (▼) will appears along with an outline of the selected item. The selected item can then be "dragged", i.e., you can hold down the mouse and move it to the window of another object. "Drop in" the selected item into the appropriate sub-window or editable box and the corresponding changes will be made. If a parameter value is dragged and dropped into another parameter box, that value will be become the new value. Likewise, an object is added to another objects list simply by dropping it in the appropriate sub-window.

Drag *compartment (2)* from the root neuron window over to the *Axial Conductance* window and drop it in the *linked compartments* sub-window. The title of the window should change to reflect the fact that the *Axial Conductance* now joins *compartment (1)* with *compartment (2)*, and the *compartment (2)* object should now appear in the *linked compartments* sub-window.



axial conductance (1:2:) {5;5}

linked compartments
[2]

compartment (1)
compartment (2)

axial conductance 7.853981634e-3 1/Ω

diameter 1.000000000e+0 μ

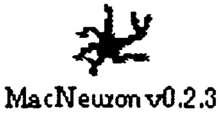
length 1.000000000e+0 μ

plasmic resistance 1.000000000e+2 Ω/cm

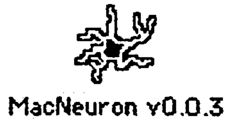
MacNeuron User's Guide

1. Startup MacNeuron program

The *MacNeuron* (version 0.2.3) program's icon looks like this:



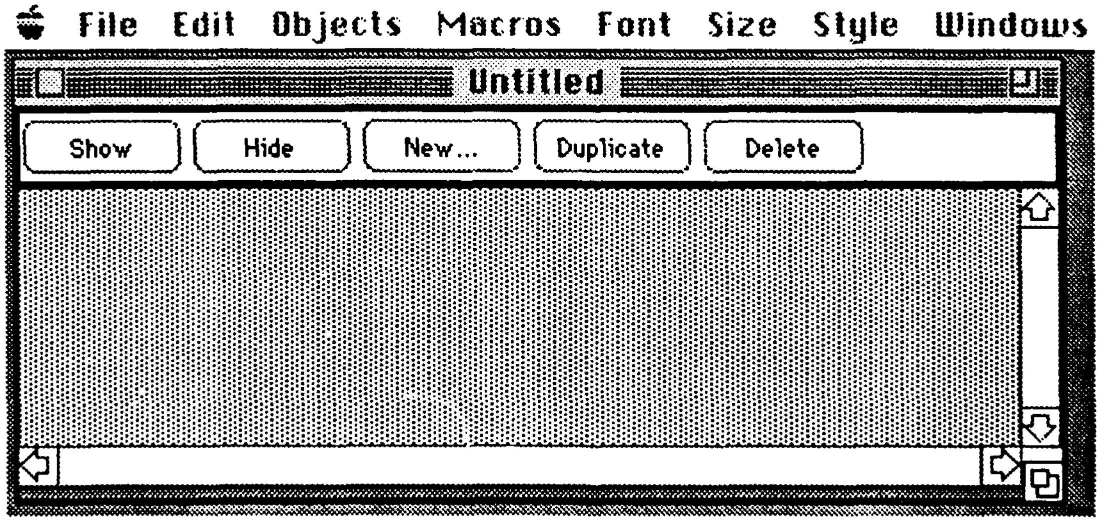
Start executing (launching) the *MacNeuron* (version 0.2.3) program by selecting the program icon:



Launch the *MacNeuron* application program as usual in the Macintosh environment by double-clicking on the icon or selecting the "*Open...*" menu-item from the "*File*" menu.

2. Main MacNeuron Window

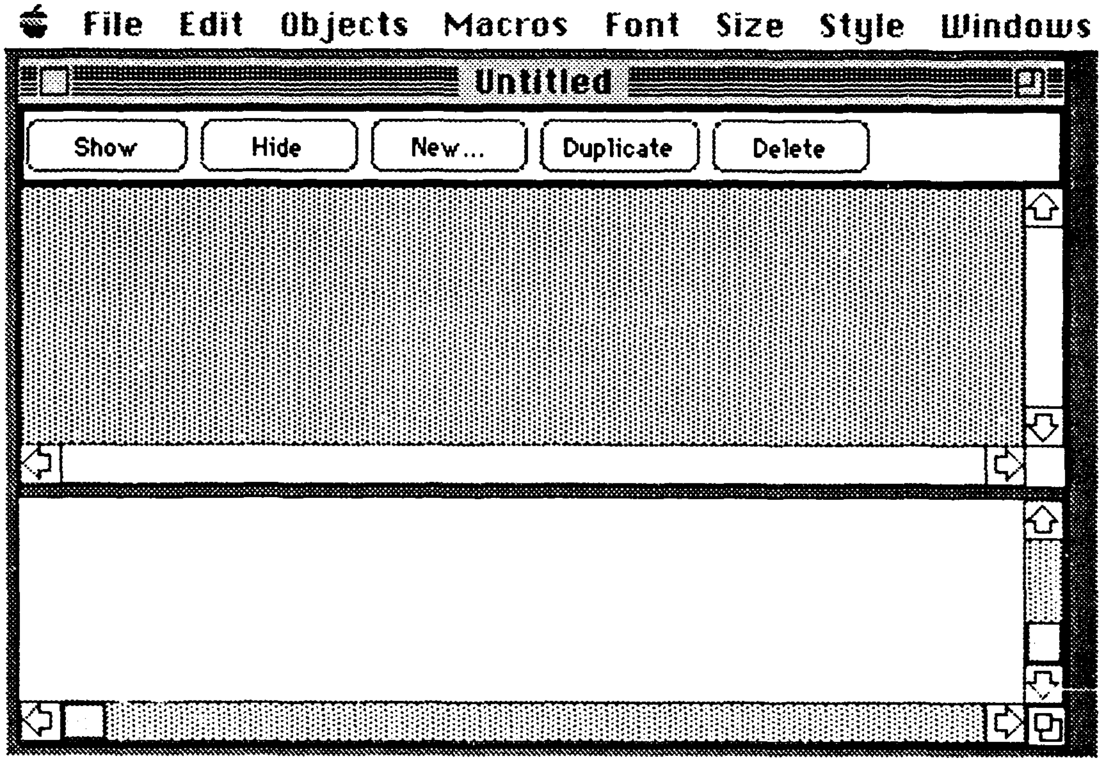
When the program first starts looks like the following :



The program will startup with a "*New*" window called "*Untitled*". This is the *main window* of the application. The name of this main window will change when the file is saved, just like any other standard Macintosh program.

2.1. Resizing the Main MacNeuron Window Partitions

This main window contains two parts (or partitions). When the resize box at the lower right corner of the window is enlarged, the second half (partition) of the window will show up. The window will appear as follows:



Alternatively, one can hold-down the mouse button at the window-partition divider (the shaded thick line between the two partitions) and "drag" the divider up or down to change the size of the partition. Initially, the partition divider is at the bottom of the window, but it can be dragged up to show the lower window partition.

2.2. Neuronal Network Description. User-Interface Window Partitions

The upper partition of this main window lists all the objects (neurons, conductances, compartments, networks, etc...) currently in the simulation. The upper portion of the window also provides a palette of tools for performing useful operations on the listed items. For instance, selecting a particular neuron from the list and then clicking the show button will cause the window for that neuron to be displayed. There are also tools for hiding, deleting and duplicating any of the objects in the list. Procedures for building the simulation will be described later in User's Guide.

2.3. Neuronal Simulation Language Description. Text-Window Partitions

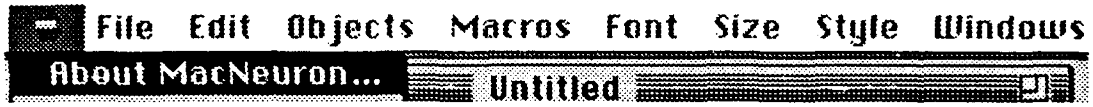
The lower partition of the main window is the text window where the simulation language is specified. A text-based language, called "*macro*", can be used to specify the simulation without recourse to the iconic user-interface. This allows the simulation to be built in a "batch-mode" or hands-off modality.

This window is also a text-editor, where the macro language description can be entered directly into the *MacNeuron* program. It is not necessary to exit the *MacNeuron* application to write the description.

The details of the macro language syntax for specifying the run-time environment will be given later in the User's Guide.

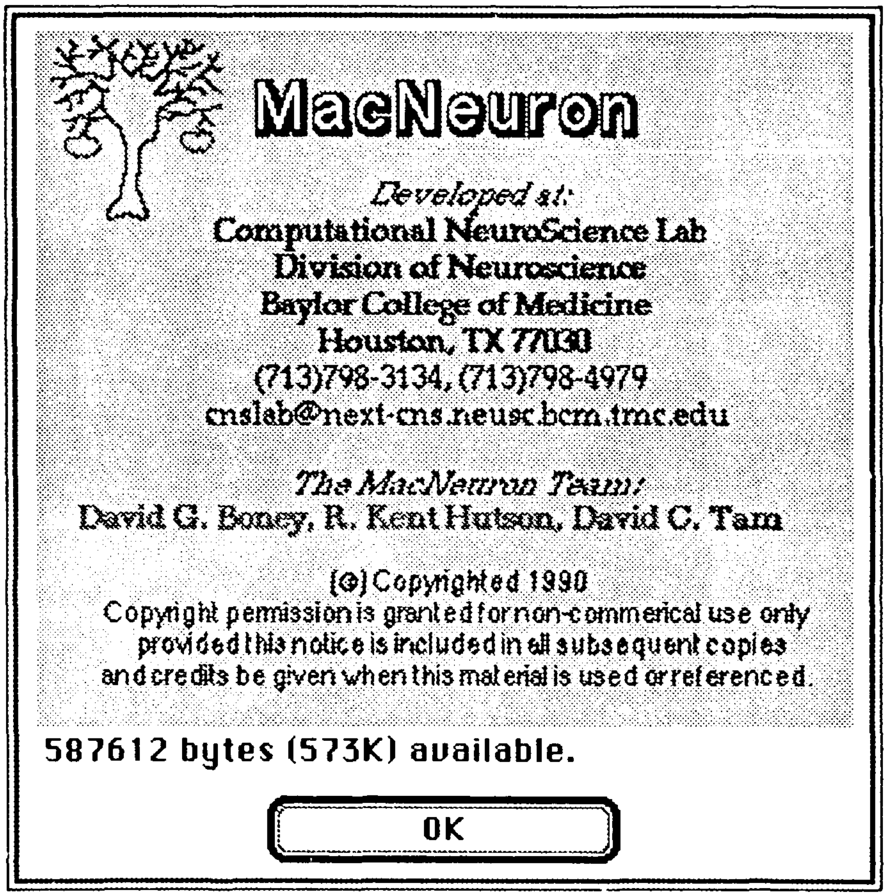
3. Main Menu Description

3.1. *Apple* Menu Description



Pulling-down the Menu will show the "*About MacNeuron...*" as the first item.

The "*About MacNeuron...*" will display the program logo as well as the available memory space for the program to run at the bottom of the logo window.



MacNeuron

Developed at:
Computational NeuroScience Lab
Division of Neuroscience
Baylor College of Medicine
Houston, TX 77030
(713)798-3134, (713)798-4979
cnslab@next-cns.neusc.bcm.tmc.edu

The MacNeuron Team:
David G. Boney, R. Kent Hutson, David C. Tam

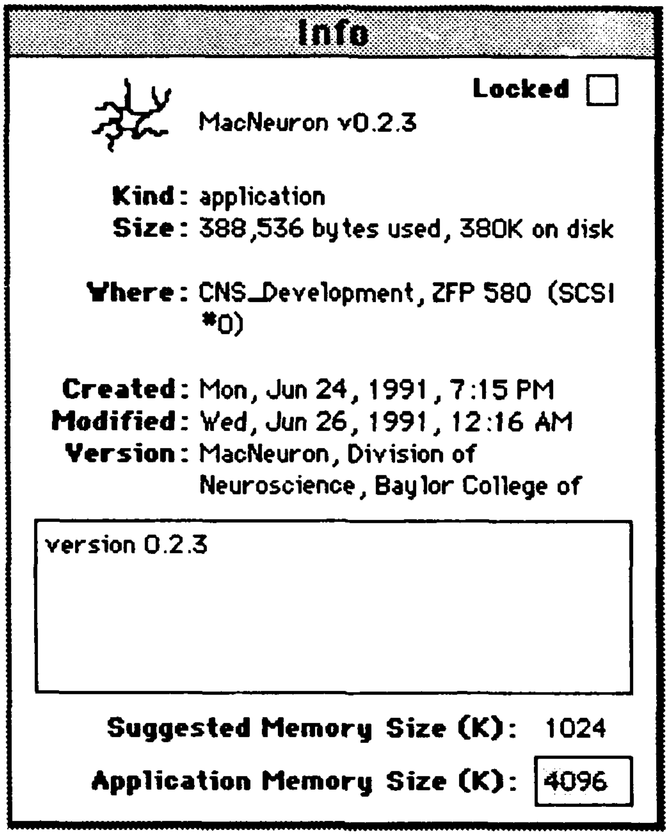
(c) Copyrighted 1990
Copyright permission is granted for non-commerical use only
provided this notice is included in all subsequent copies
and credits be given when this material is used or referenced.

587612 bytes (573K) available.

3.1.1. Memory Space Available and Allocation

To find out how much memory is available at anytime while inside the simulator environment, select the "*About MacNeuron...*" from the *Apple* Menu as shown above. The memory available depends on the complexity of the neuronal network specified and other run-time user-interface environments.

To increase or decrease the size of the memory allocation, quit the program. Select the *MacNeuron* program icon under *Finder,* and select the "*Get Info*" menu-item from the *File* Menu. An *Info* window similar to the one below will be displayed.



Info

Locked □

MacNeuron v0.2.3

Kind: application
Size: 388,536 bytes used, 380K on disk

Where: CNS_Development, ZFP 580 (SCSI #0)

Created: Mon, Jun 24, 1991, 7:15 PM
Modified: Wed, Jun 26, 1991, 12:16 AM
Version: MacNeuron, Division of Neuroscience, Baylor College of

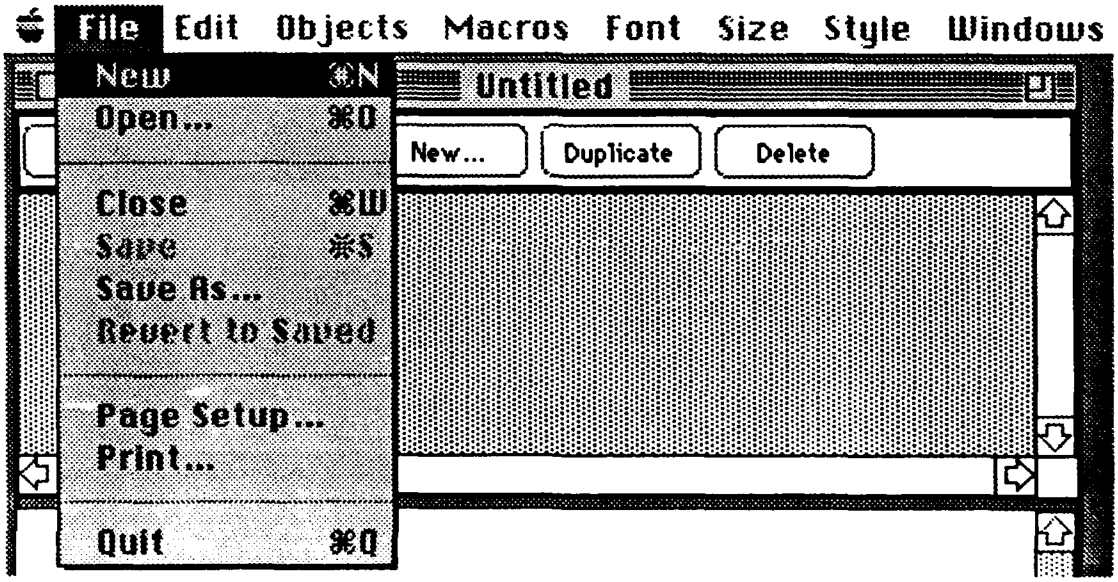
version 0.2.3

Suggested Memory Size (K): 1024

Application Memory Size (K): 4096

Change the size of the *MacNeuron* program by selecting the *Application Memory Size (K)* box and edit the changes in Kilobytes.

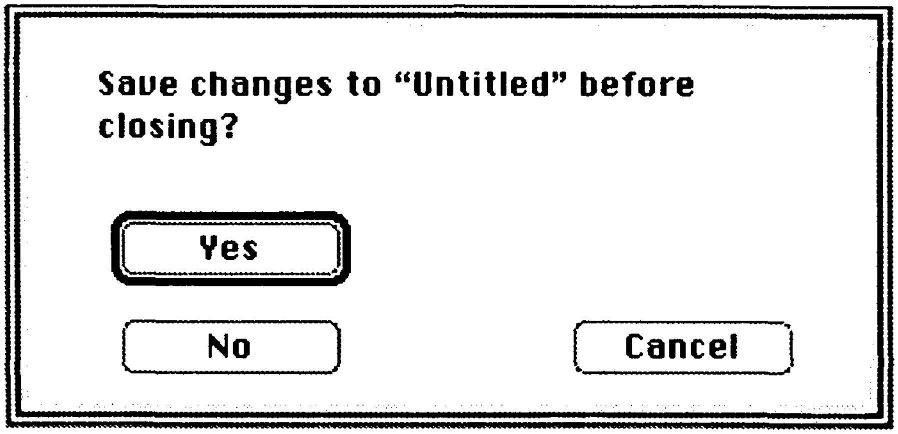
3.2. *File* Menu Description



The *File* Menu has its usual standard Macintosh-menu items for file operations such as opening and closing files, saving and printing files, creating new files, quitting the application, etc.

When the program is first launched, a new file is created automatically for the user and appears as the "*Untitled*" main *MacNeuron* description window. This would also be the consequence of clicking the "*New*" menu-item in the *File* Menu.

To close this main window, select the "*Close*" menu-item or click on the *close-*box of the window at the upper-left corner of the window as usual. A warning dialog-box will be displayed is you have not saved the file before to make sure that you don't loose any valuable work.


Save changes to “Untitled” before closing?

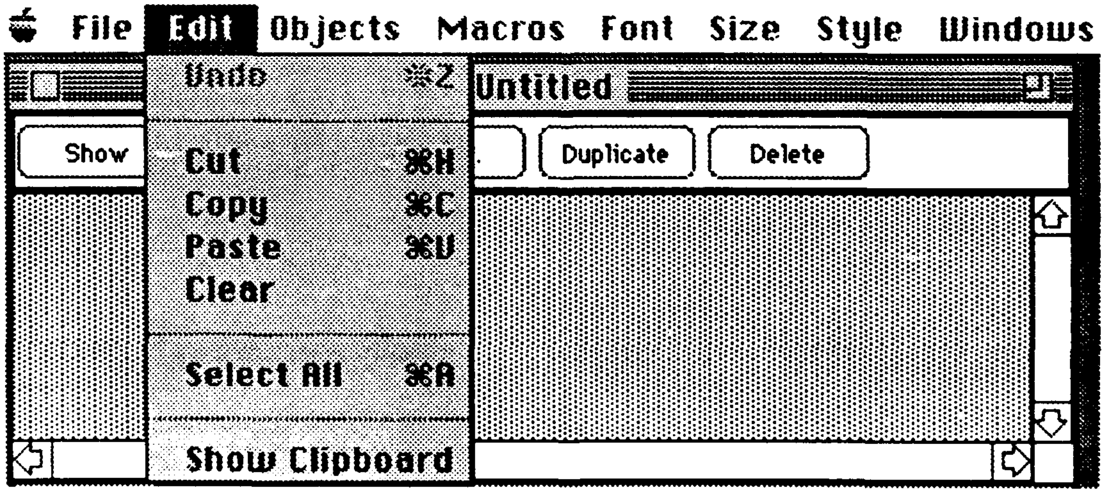
Yes

No

Cancel

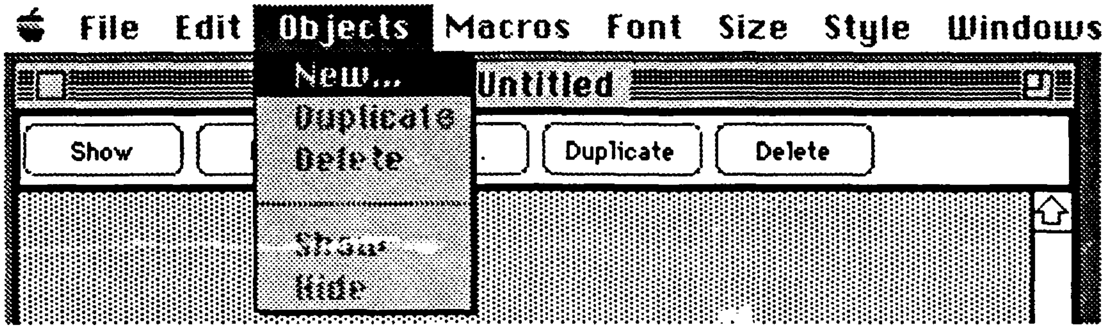
Select the appropriate button or press the Return-key of the keyboard to select the highlighted button, which is "*Yes*" in this case.

3.3. *Edit* Menu Description



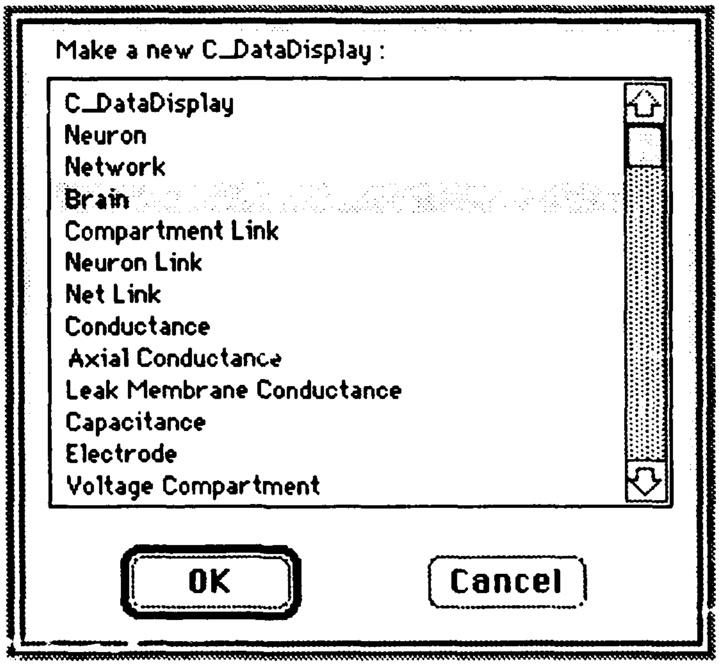
The *Edit* Menu has its usual standard Macintosh-menu items for editing. The *Cut, Copy, Paste* and *Clear* menu items can be used with the text window for editing the simulation macro description language (which is a lower sub-window in the main window).

3.4. *Objects* Menu Description



Objects in the *MacNeuron* program are the implementations of the components of a neuron or a network. The currently available objects in *MacNeuron* include *brain, network, neuron, voltage compartment, conductance, axial conductance, membrane conductance, leakage conductance, capacitance, electrode, active conductance, HH gate* (Hodgkin-Huxley), *compartment link, neuron link, network link,* etc. as well as objects for displaying data.

A list of the available objects will be displayed in a dialog box when the "*New...*" menu item from the *Objects* Menu is selected.


Make a new C_DataDisplay :

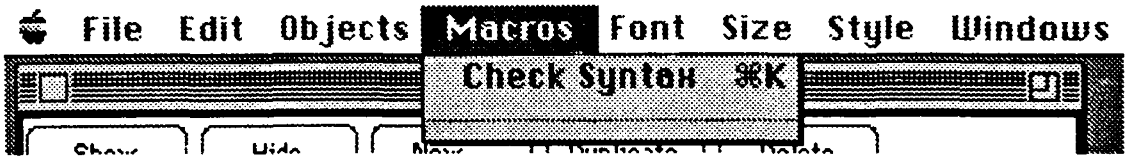
The other menu items, such as *Duplicate, Delete, Show* and *Hide* are dimmed initially since no object exists until a new objects is created first.

3.4.1. Creating New *Objects* from the *Objects* Menu

To create new objects, select the desired object in the above dialog box, and click the "*OK*" button. Alternatively, you can double-click on the selected object without clicking on the "*OK*" button to create that new object. To abort the creation of a new objects, click the "*Cancel*" button.

Once the new object is created or "*New*ed", a new window associated with that new object will pop up. In the above example, since the *brain* object is selected for creation, the *brain* window will pop up as the front window. The detailed descriptions and utilities of this new object window will be discussed later in this User's Guide.

3.5. *Macros* Menu Description



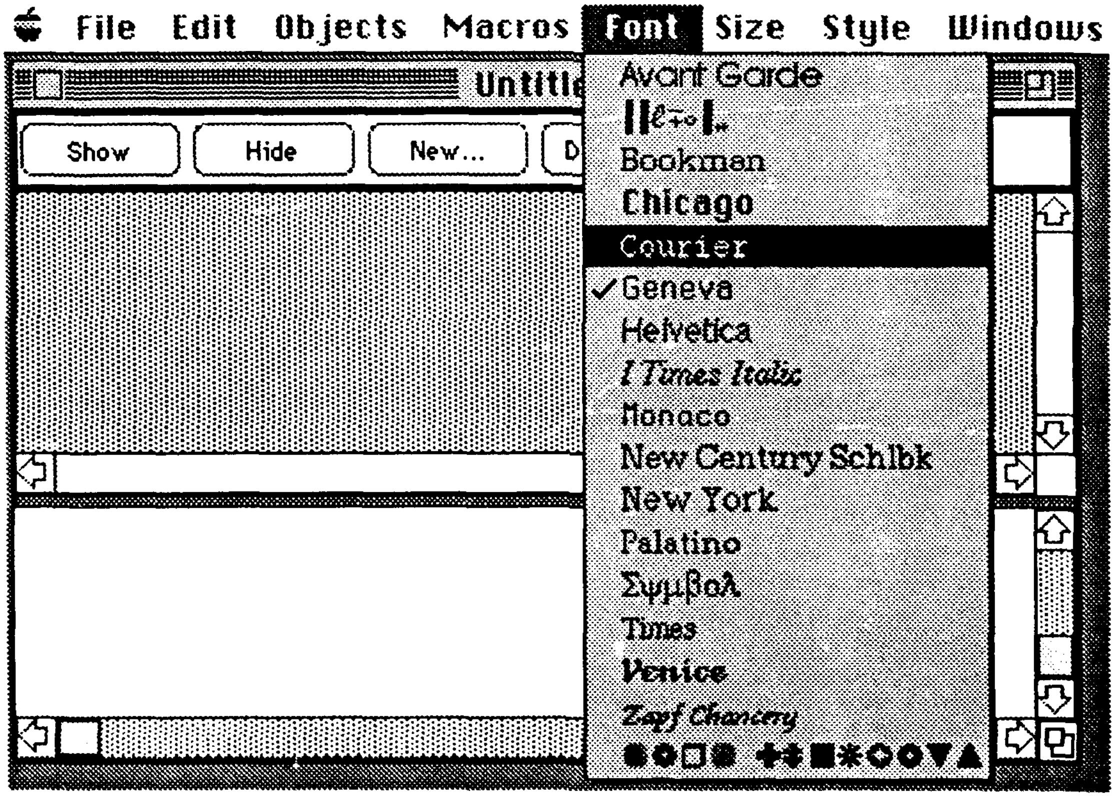
Macros are groups of commands specified by the user in the simulation macro description language. The simulation description language—*MacNeuron Script*—is a Pascal-like language that allows a hands-off specification of the simulation bypassing the iconic user-interface. Such a feature is very useful for constructing large simulations intended to run in a batch or background mode. Details of the language syntax will be given later in the User's Guide.

If the text of *MacNeuron Script* is entered in the text-editor window at the lower partition of the main window, selecting the "*Check Syntax*" menu item from the *Macros* Menu will check the syntax of the *MacNeuron Script* entered by the user. If there are any syntax errors, the appropriate error message will be displayed in a warning dialog box. The location of the offending error will be highlighted in the text-editor window to allow for easy recognition by the user.

Once the syntax has been checked, and there are no syntax errors, the *macros* defined in the *MacNeuron Script* will be displayed as extra menu items appended to the end of the *Macros* Menu lists. *Macros* are basically groups of commands that the user specifies in the *MacNeuron Script* so that those groups of commands can be executed as a menu command available in the menu-item list. The user can therefore create his/her own menu-item list on-the-fly within the *MacNeuron* user environment.

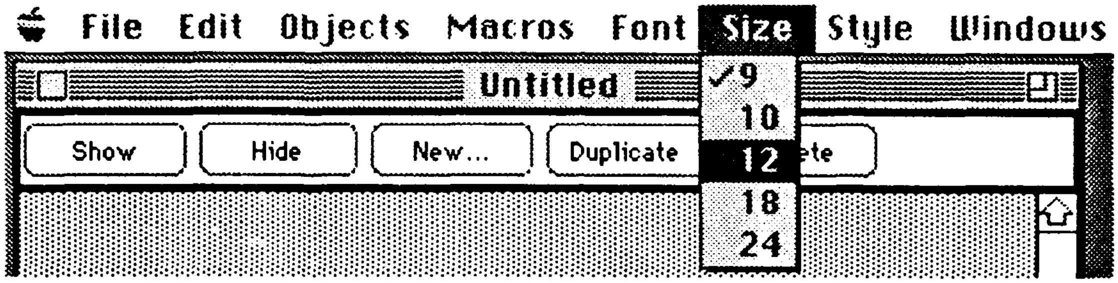
For instance, if the user groups a set of commands that specify the description of all the compartments of a Purkinje neuron into a single *macros* command called "*Create Purkinje Cell*", then a Purkinje cell can be created with just a mouse click from the *Macros* Menu. Thus, multiple Purkinje cells can be created easily by "pulling down" the *Macros* Menu and selecting the "*Create Purkinje Cell*" menu item that was defined by the user in the *MacNeuron Script* in the text-editor window.

3.6. *Font* Menu Description



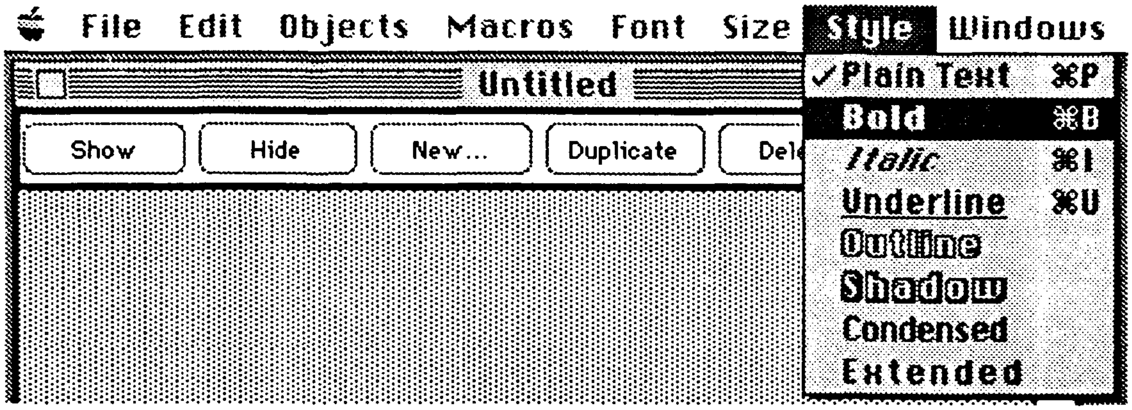
The *Font* selected by this menu will alter the font used in the text-editor sub-window of the main *MacNeuron* window. The available fonts are the fonts that are installed in the System file of your Macintosh disk. The actual font-type displayed in the *Font* Menu above is generated by a utility program called Suitcase™ installed in the System, otherwise the system-font will be used to display the various fonts available. *Geneva* is the default font.

3.7. *Size* Menu Description



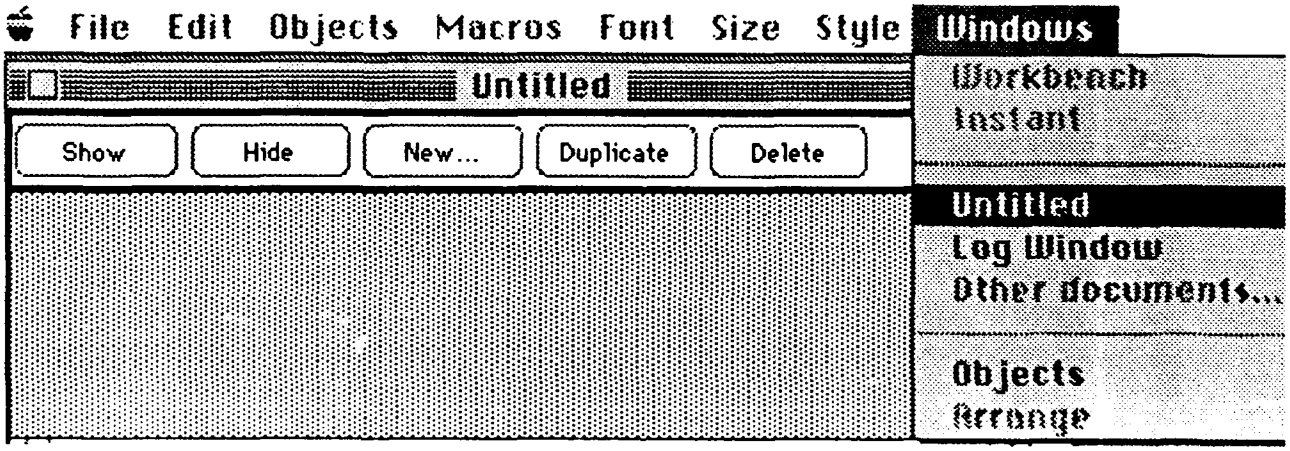
The *Size* selected by this menu will alter the font size used in the text-editor sub-window of the main *MacNeuron* window. The default font size is 9 points.

3.8. *Style* Menu Description



The *Style* selected by this menu will alter the font style used in the text-editor sub-window of the main *MacNeuron* window. *Plain-Text* is the default style.

3.9. *Windows* Menu Description



The *Windows* Menu lists all the available windows, and allows all of them to be selected as the active window (the front window).

3.9.1. *Main Window* Menu-Item

The *"Untitled"* window is the main *MacNeuron* window if it has not been saved before. If an existing file is opened, the name of that file is displayed in that menu-item location. Selecting it will make that window active (it will appear as the front window).

3.9.2. *Log Window* Menu-Item

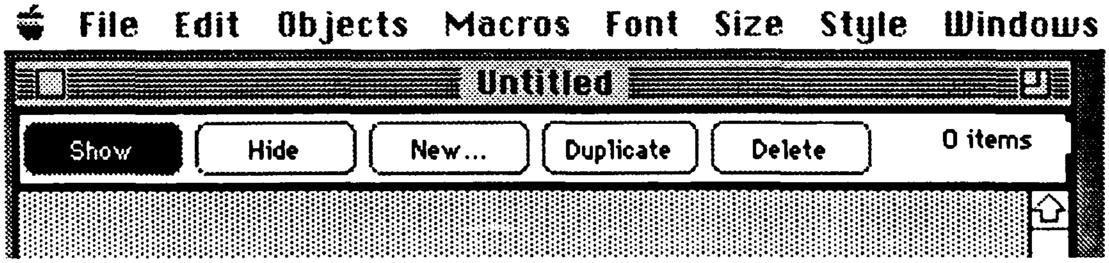
The *Log Window* is a text display window where the program keeps a log of the history of the simulation. The user can also select specific parameters to be output into this *Log Window*. Selecting it will make that window active (it will appear as the front window).

If the program is run in unattended batch-mode, the specified simulation textual output and any error messages will be displayed in this *Log Window*. If there are critical errors that require user's intervention (such as File-Not-Found) while running in batch-mode, the program will wait for a "time-out" period. If no user's action is taken after the time-out, a default value (such as a default file-name) will be used. This will allow for unattended continued simulation over-night without creating halting the program waiting for the user's response.

3.9.3. *Object Window* Menu-Item

The *Object Window* is a window where the contents (and the parameters) of an *object* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, if the object is a neuron, then the contents of that neuron, i.e., its parameters will be displayed in this object window.

4. *Main MacNeuron User-Interface Description Window*



The main *MacNeuron* user-interface description window (the upper partition of the Main window) contains five buttons. These five buttons are the same as the menu items listed under the *Object* Menu (see *Object* Menu Description above). Their use is interchangeable. That is, the buttons function the same as the menu items in the *Object* Menu.

4.1. *Show* Button

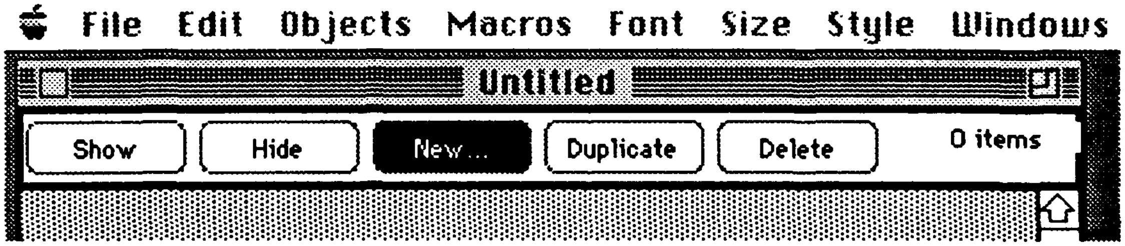
When the "*Show*" button is pressed, it will show the object that is selected in a separate *Object Window*. Since, at this point, no objects are created yet, there is nothing to show. The "*0 items*" is indicated at the upper-right of the window to show the number of object-items created so far.

Once objects are created, they will be listed in the shaded region of the window. Select the object by clicking on the item in the list, and click the "*Show*" button. The *Object Window* associated with that selected object will pop up (open). The contents of that object (i.e., its parameters) will be displayed in the *Object Window*.

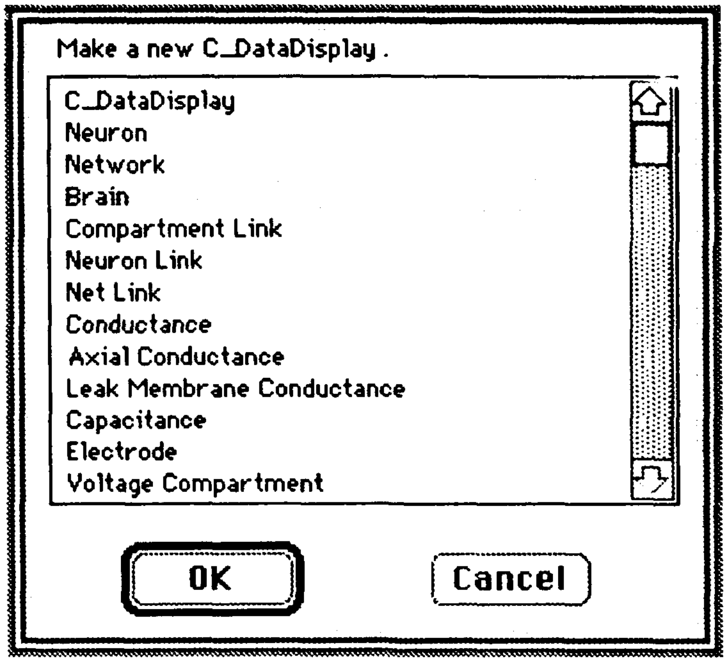
4.2. *Hide* Button

When the "*Hide*" button is pressed, it will hide (close) the *Object Window*. Which *Object Window* will be closed depends upon the selected objects in the object list displayed in this window.

4.3. *New...* Button



When the "*New...*" button is pressed, a list of the available objects will be displayed in a dialog box. Select the object to be created as described above (see the Creating New *Objects* from the *Objects* Menu Section).



Make a new C_DataDisplay .

C_DataDisplay
Neuron
Network
Brain
Compartment Link
Neuron Link
Net Link
Conductance
Axial Conductance
Leak Membrane Conductance
Capacitance
Electrode
Voltage Compartment

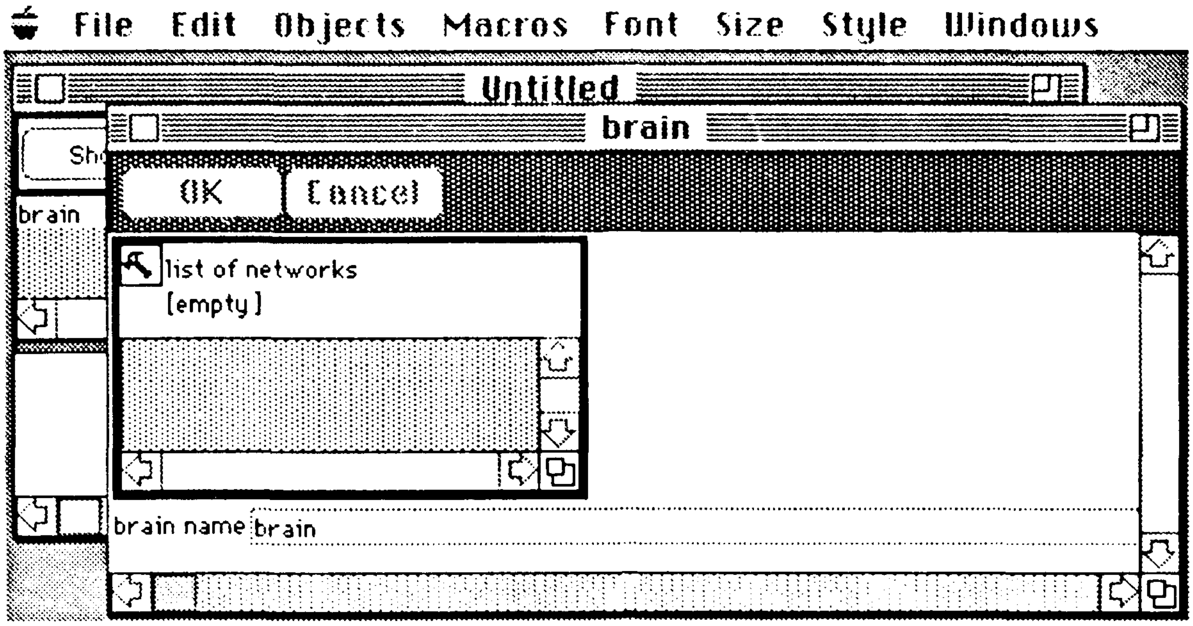
4.4. *Duplicate* Button

When the "*Duplicate*" button is pressed, the selected objects in the object list displayed in this window will be duplicated.

4.5. *Delete* Button

When the "*Delete*" button is pressed, the selected objects in the object list displayed in this window will be deleted.

5. *Brain Object* Window



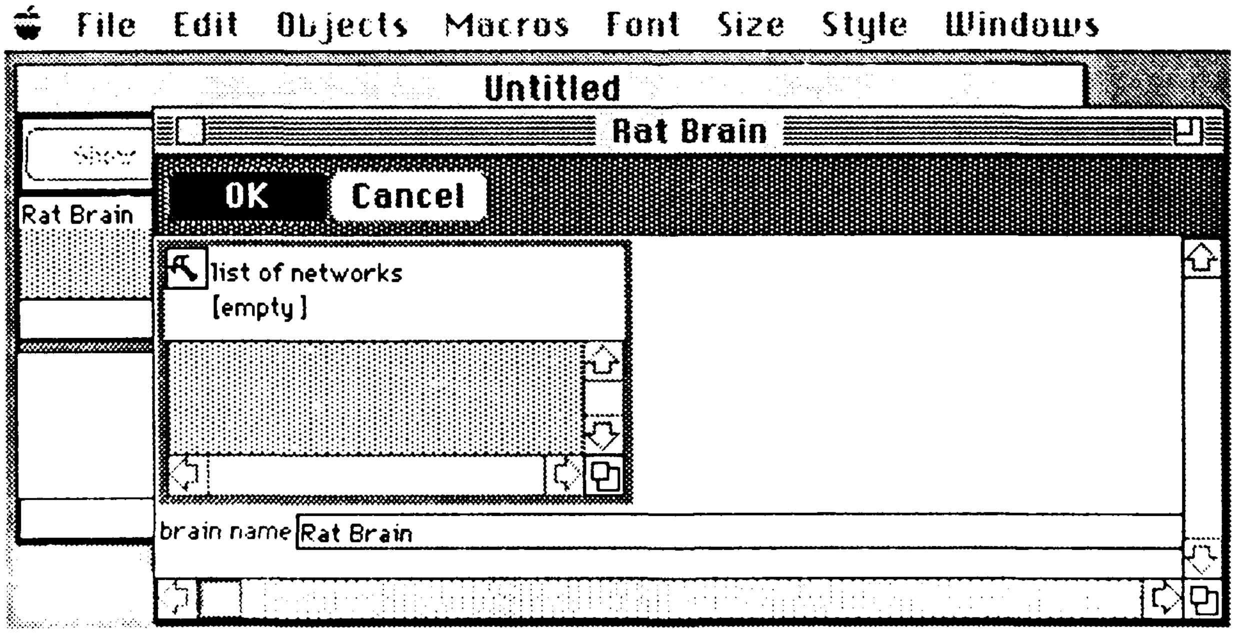
brain

list of networks
[empty]

brain name brain

The *Brain Object* window is a window where the contents (and parameters) of the *brain* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, since in this case the object is a *brain*, the content of that *brain*, i.e., its parameters, will be displayed in this object window. The *title* of this *Brain Object* window will be called by the name of the object, i.e., the *brain* by default. The name of the *Brain Object* can be changed, as it will be discussed later.

5.1. *List of Network Objects* Window



Rat Brain

list of networks
[empty]

brain name Rat Brain

There are usually two sets of items to be displayed in a *Brain Object* window. The first set of items is the "list of other objects under its hierarchy". In this example, the ***brain*** is composed of a list of ***networks***. Similarly, a ***network*** is composed of a list of ***neurons***. A ***neuron*** is composed of a list of ***compartments***, etc. Thus, the anatomical structure of a brain, a network, a neuron, etc., can be specified hierarchically.

In this example, since the ***brain*** has just been created, the list of ***networks*** is not specified yet. So "*[empty]*" is indicated in the "*list of networks*" sub-window.

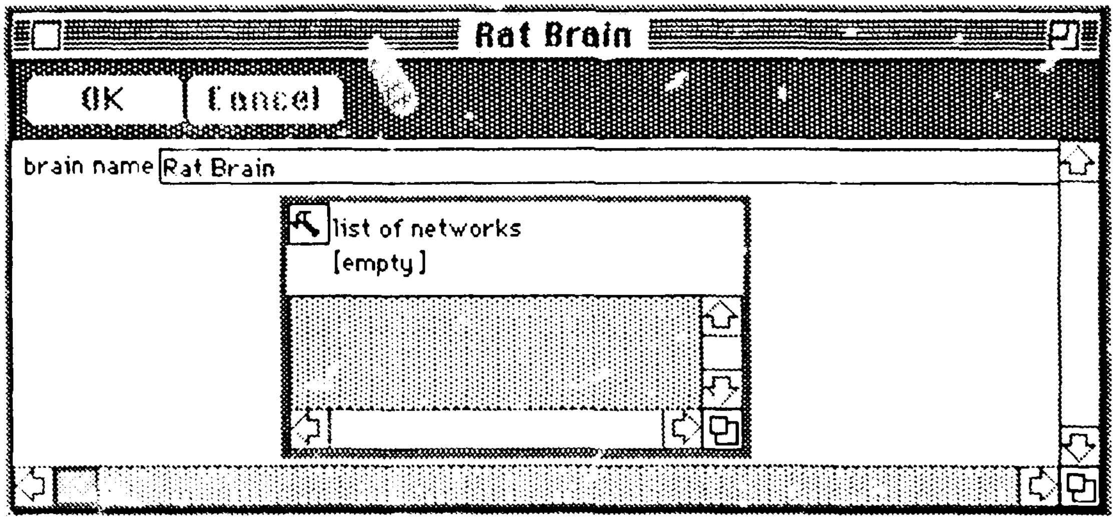
Note that the "*list of networks*" sub-window is highlighted (i.e., a thick dark square outlines the window. This is the item which is currently selected. You can change the selected item by pressing the "tab" key in the keyboard to "tab over" to the next field as in any Macintosh application environment. The next item is the "***brain name***" which will be highlighted.

Alternatively, you can use the mouse to move the pointer over the "***brain name***" box and edit the text as usual. The "***brain name***" box will be selected (or highlighted), and the text can be changed accordingly.

In the example shown above, the name is changed into "*Rat Brain*". Press the "*OK*" button to confirm the changes or press the "*Cancel*" button to cancel and revert the changes. When the "*OK*" button is press, the new name "*Rat Brain*" will be reflected in the title bar of the current "***Brain***" window as the new "*Rat Brain*" window. This new brain name is also reflected in the item list of the main window, which is still called "*Untitled*" presently (half-hidden behind the "*Rat Brain*" window in the above example).

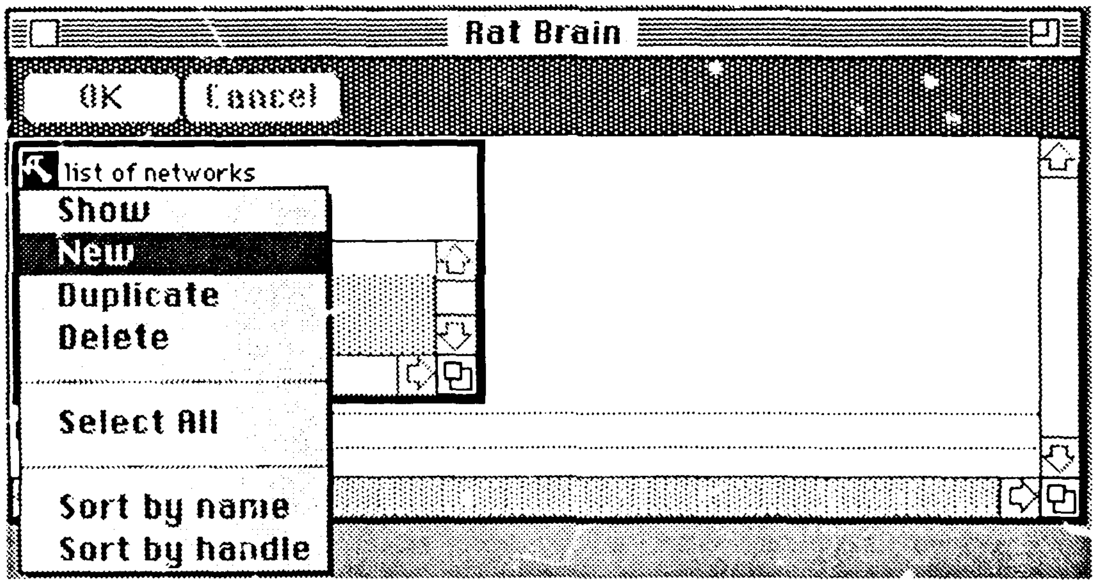
Note that the appearance of the layout arrangement of the selectable items such as the "*list of networks*" sub-window and the "*brain name*" box can be rearranged by the user. Holding down the command-key (or the "apple-clover" key) of the Macintosh keyboard while pressing the mouse button over the selected item will change the cursor into a four-arrow sign (✥), indicating that the user can now move that item to a new location within the window. This four-arrow sign will also appear when the cursor is placed above the name field of the "*list of networks*" sub-window without holding down the command key.

For instance, the layout of the "*Rat Brain*" window can be re-arranged to look like the following by the user:



brain name Rat Brain

5.1.1. Pop-up Menu in the *List of Networks* Window



When the *tool* icon (🔨) at the upper-left corner of the *List of Objects* sub-window is pressed, a pop-up menu will appear as described below.

5.1.2. *Show* Menu-Item from the Pop-up Menu

The *Show* Menu-Item shows the selected network from the list of networks created so far. A *Network Object* window will pop-up displaying the contents (parameters) of the network.

Alternatively, the selected item (from the lists of networks) can be double-clicked to show the *Network Object* window.

5.1.3. *New* Menu Item from the Pop-up Menu

The *New* Menu Item creates a new network and numbers it sequentially with an integer number in parenthesis. The new network will be displayed in the shaded region of the list cf networks window.

5.1.4. *Duplicate* Menu Item from the Pop-up Menu

The *Duplicate* Menu Item clones a new network and numbers the newly cloned network sequentially with an integer number in parenthesis. The new network will be displayed in the shaded region of the list of networks window.

5.1.5. *Delete* Menu Item from the Pop-up Menu

The *Delete* Menu Item removes a selected existing network from the list of networks displayed in the shaded region of the list of networks window.

5.1.6. *Select All* Menu Item from the Pop-up Menu

The *Select All* Menu Item selects all the existing networks from the list of networks displayed in the shaded region of the list of networks window.

5.1.7. *Sort by name* Menu Item from the Pop-up Menu

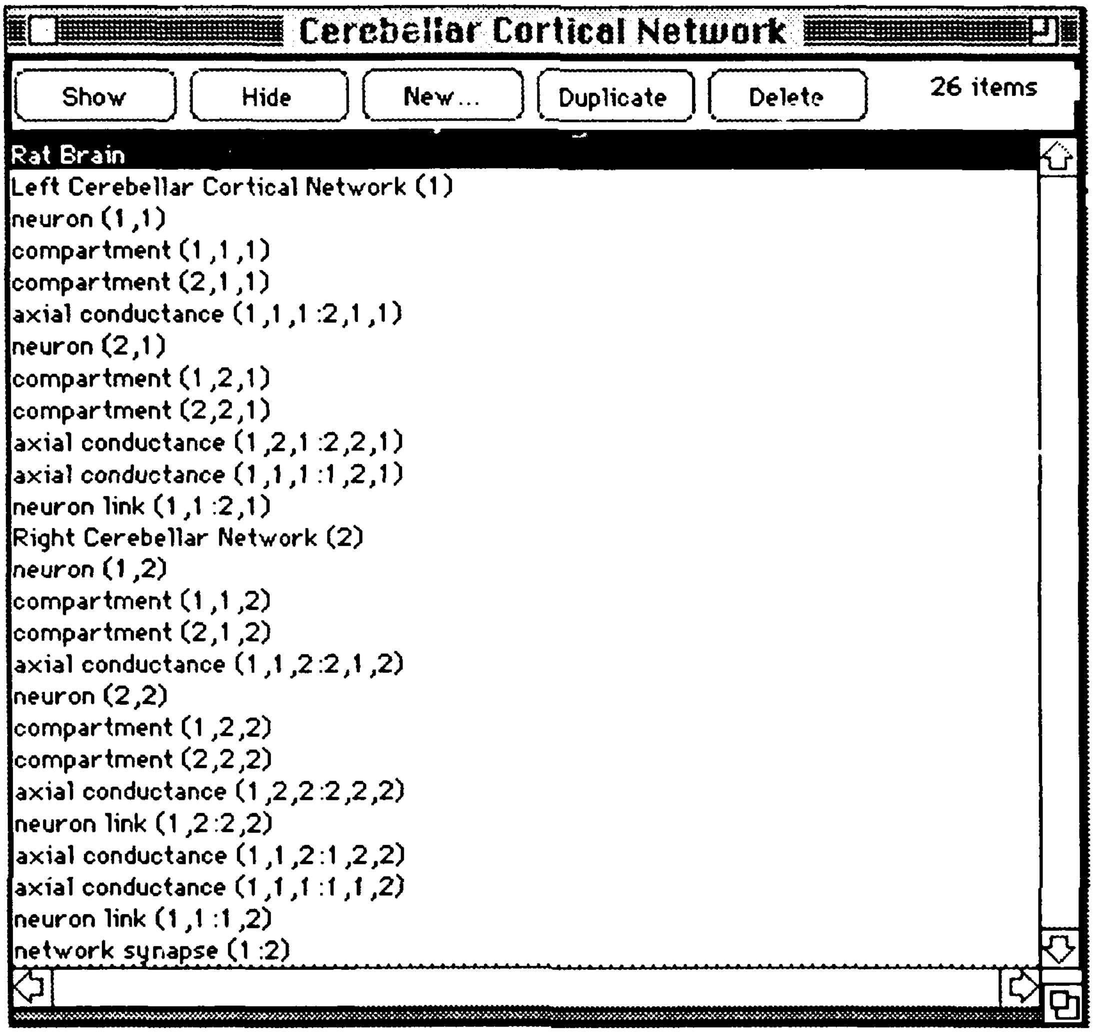
The *Sort by name* Menu Item re-orders the existing networks in alphabetical order using the name of the network displayed in the shaded region of the list of networks window.

5.1.8. *Sort by handle* Menu Item from the Pop-up Menu

The *Sort by handle* Menu Item re-orders the existing networks in internal order managed by the computer of the network (called the handle to the network).

6. Cerebellar Cortical Network Example

We will use the pre-built Cerebellar Cortical Network file as an example.



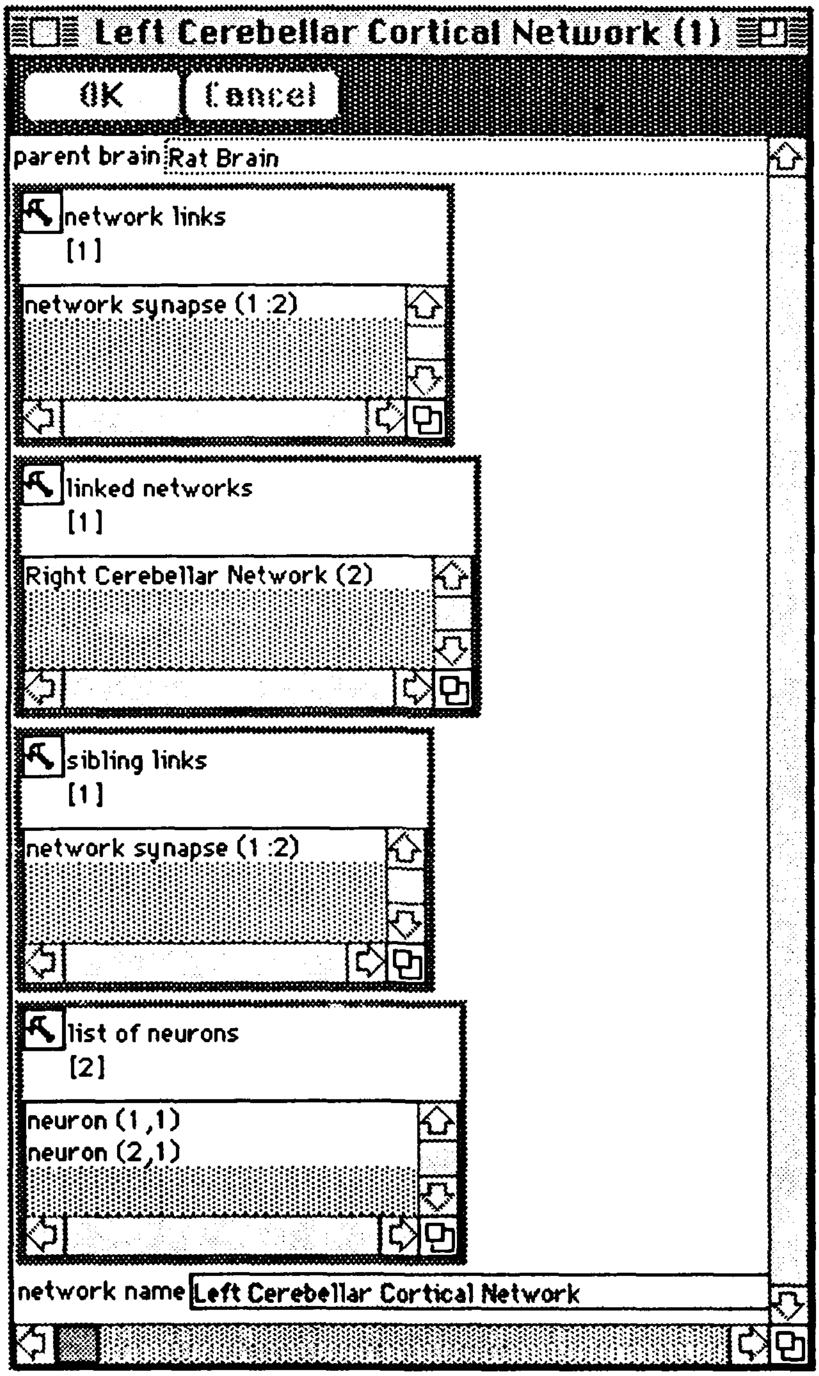
```
Rat Brain
Left Cerebellar Cortical Network (1)
neuron (1,1)
compartment (1,1,1)
compartment (2,1,1)
axial conductance (1,1,1:2,1,1)
neuron (2,1)
compartment (1,2,1)
compartment (2,2,1)
axial conductance (1,2,1:2,2,1)
axial conductance (1,1,1:1,2,1)
neuron link (1,1:2,1)
Right Cerebellar Network (2)
neuron (1,2)
compartment (1,1,2)
compartment (2,1,2)
axial conductance (1,1,2:2,1,2)
neuron (2,2)
compartment (1,2,2)
compartment (2,2,2)
axial conductance (1,2,2:2,2,2)
neuron link (1,2:2,2)
axial conductance (1,1,2:1,2,2)
axial conductance (1,1,1:1,1,2)
neuron link (1,1:1,2)
network synapse (1:2)
```

7. *Network Object* Window

The *Network Object* window is a window where the contents (and parameters) of the *network* are displayed. Selecting it will make that window active (it will appear as the front window). For instance, since in this case the object is a *network,* the content of that *network,* i.e., its parameters will be displayed in this object window. The *title* of this *Network Object* window will be the name of the object, i.e., the *network (1)* by default. The name of the *Network Object* can be changed, as it will be discussed later.

Since the simulated neural network is constructed hierarchically, we will use family-tree terminology to refer to the hierarchical structure, such as parent object, sibling links, etc.

In this example, the parent object of this newly created network is the *Rat Brain*. The name of this network can be changed to another name, say *Cerebellar Cortical Network*.



parent brain Rat Brain

network links
[1]
network synapse (1 :2)

linked networks
[1]
Right Cerebellar Network (2)

sibling links
[1]
network synapse (1 :2)

list of neurons
[2]
neuron (1,1)
neuron (2,1)

network name Left Cerebellar Cortical Network

7.1. *Network Links* Sub-Window

This shows the list of network synapses to which the current network is connected. The numeric indices in parentheses refer to the network and the network synapse by their number (i.e., their name for identity). It shows that *Left Cerebellar Network (1)* is connected to the *Right Cerebellar Network (2)* by specifying *Network Synapse (1:2)*.

7.2. *Linked Networks* Sub-Window

This shows the list of connected networks. It is connected to the *Right Cerebellar Network (2)*.

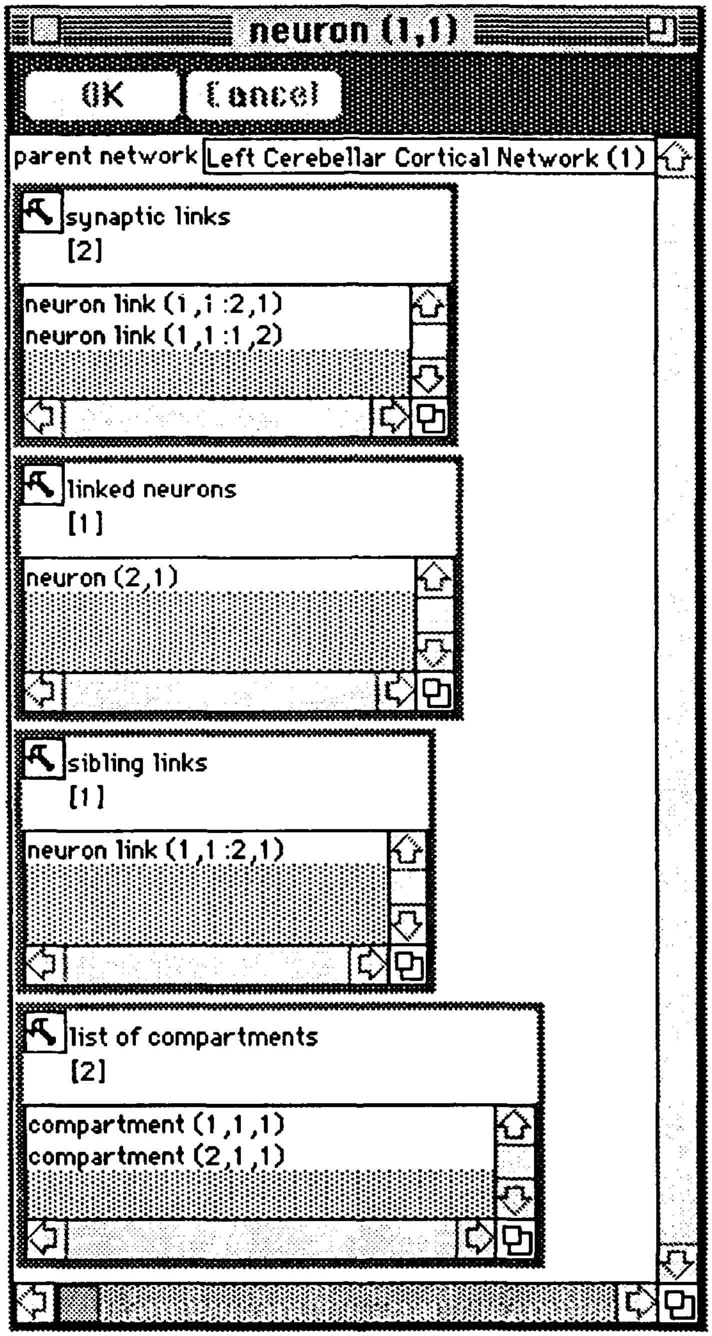
7.3. *Sibling Links* Sub-Window

This shows the list of sibling networks that it is connected to. Its sibling link is *Network Synapse (1:2)*.

7.4. *List of Neurons* Sub-Window

This shows the list of neurons that it is connected to. It has two neurons: *Neuron (1,1)* and *Neuron (2,1)*.

8. *Neuron Object* Window



parent network Left Cerebellar Cortical Network (1)

synaptic links
[2]

neuron link (1,1 :2,1)
neuron link (1,1 :1,2)

linked neurons
[1]

neuron (2,1)

sibling links
[1]

neuron link (1,1 :2,1)

list of compartments
[2]

compartment (1,1,1)
compartment (2,1,1)

8.1. *Synaptic Links* Sub-Window

This shows the list of neurons to which the current neuron is connected. The numeric indices in parentheses refer to the unique neuron and network identification numbers. It shows that *neuron (1)* in the *Left Cerebellar Network (1)* is connected to the *neuron (2)* in the *Left Cerebellar Network (1)* as specified by *Neuron Link (1,1:2:1)*.

8.2. *Linked Neurons* Sub-Window

This shows the list of connected neurons. It is connected to *Neuron (2)*.

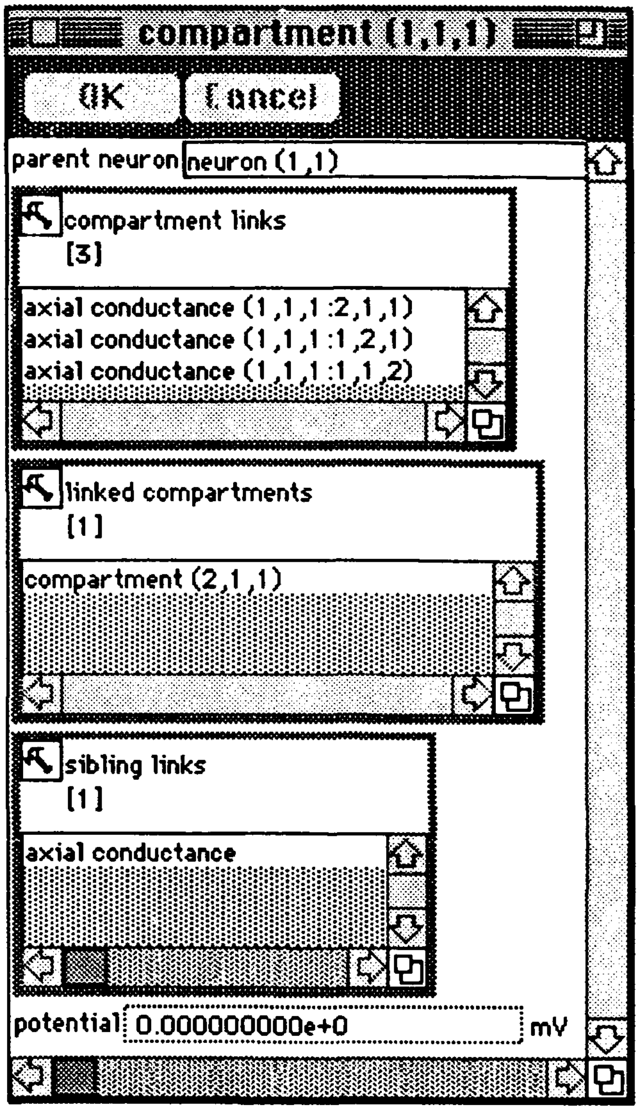
8.3. *Sibling Links* Sub-Window

This shows the list of sibling neuron to which it is connected. Its sibling link is *Neuron Link (1,1:2,1)*.

8.4. *List of Compartments* Sub-Window

This shows the list of compartments to which it is connected. It has two neurons: *Compartment (1,1,1)* and *Compartment (2,1,1)*. That is, *Compartment (1)* in *Neuron (1)* in *Left Cerebellar Network (1)* and *Compartment (2)* in *Neuron (1)* in *Left Cerebellar Network (1)*.

9. *Compartment Object* Window



parent neuron neuron (1,1)

compartment links
[3]

axial conductance (1,1,1:2,1,1)
axial conductance (1,1,1:1,2,1)
axial conductance (1,1,1:1,1,2)

linked compartments
[1]

compartment (2,1,1)

sibling links
[1]

axial conductance

potential 0.000000000e+0 mV

9.1. *Compartment Links* Sub-Window

This shows the list of axial conductances to which the current compartment is connected. The numeric indices in parentheses refer to the axial conductance by their number (i.e., their name for identity).

9.2. *Linked Compartments* Sub-Window

This shows the list of connected compartments. It is connected to the *Compartments (2)* in *Neuron (1)* in *Left Cerebellar Network (1).*

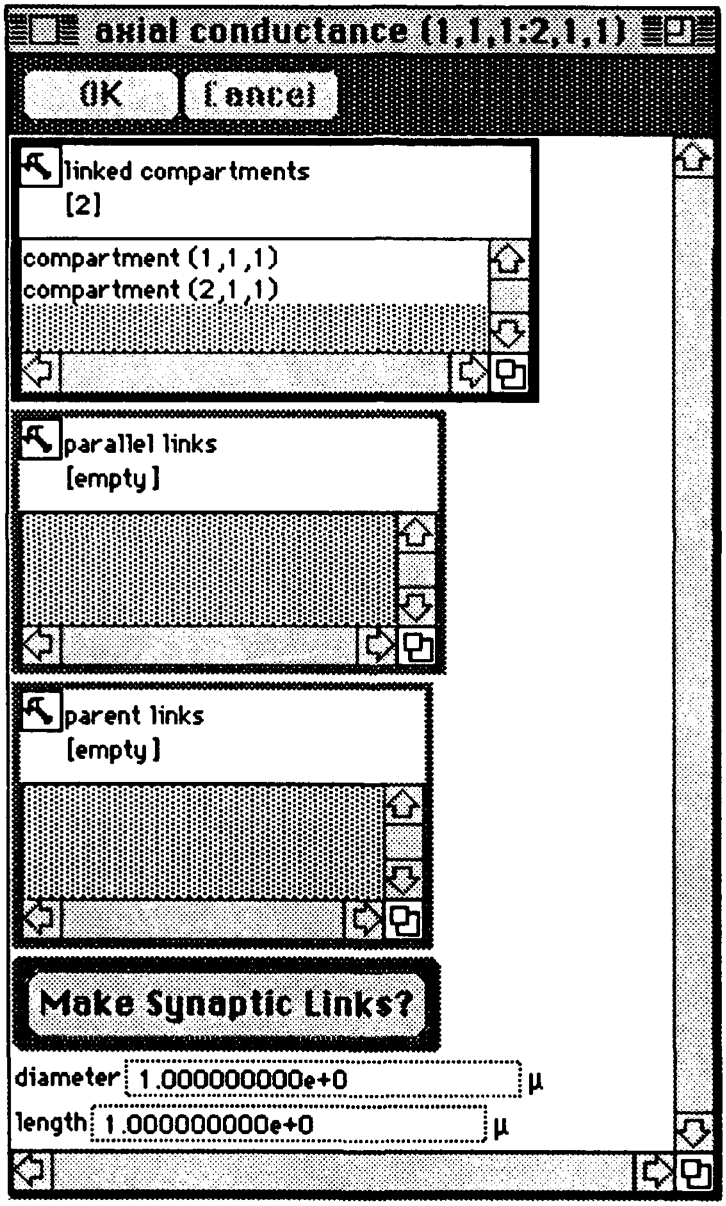
9.3. *Sibling Links* Sub-Window

This shows the list of sibling axial conductances to which it is connected.

9.4. *Potential* Box

This shows *potential* of the compartment in mV.

10. *Axial Conductance Object* Window



linked compartments
[2]

compartment (1,1,1)
compartment (2,1,1)

parallel links
[empty]

parent links
[empty]

diameter 1.000000000e+0 µ
length 1.000000000e+0 µ

Axial conductance object includes the conductance, as well as the connection between adjacent patches of compartmental membranes.

10.1. *Linked Compartments* Sub-Window

This shows the list of connected compartments.

10.2. *Parallel Links* Sub-Window

This shows the list of sibling axial conductance to which it is connected.

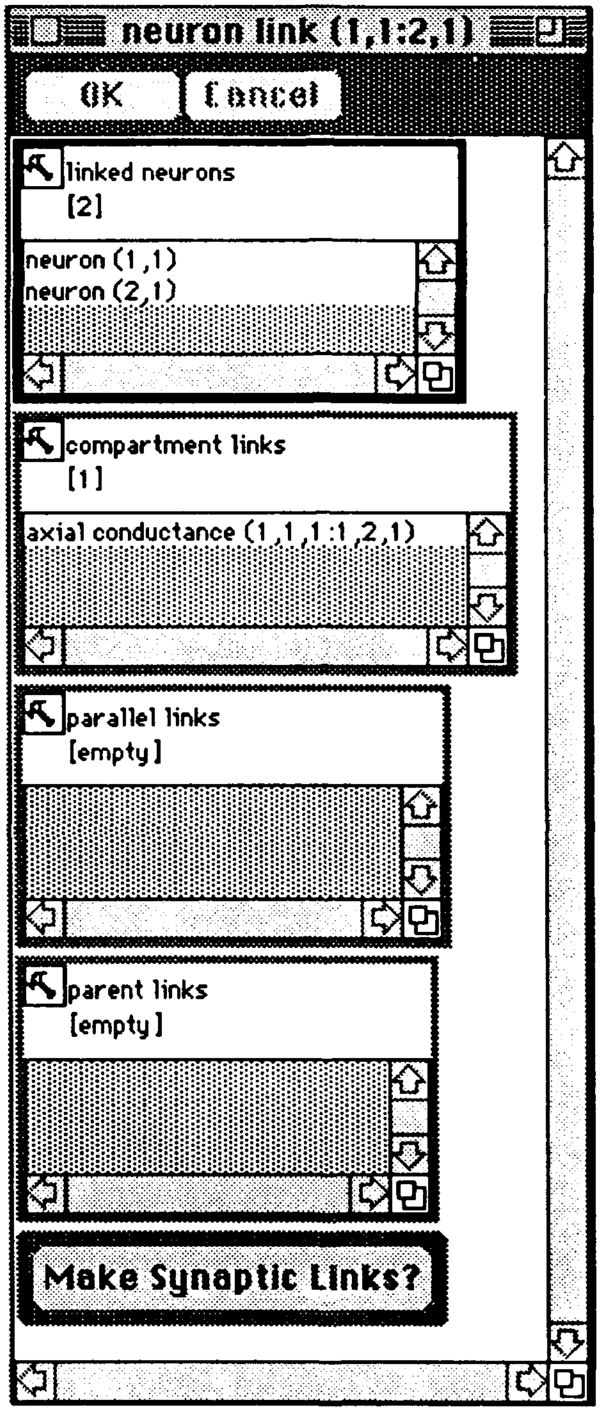
10.3. *Parent Links* Sub-Window

This shows the list of parent axial conductance to which the current compartment is connected.

10.4. *Make Synaptic Links* Button

It asks if synaptic links will be added.

11. *Neuron Link Object* Window



Neuron link object is the connection between adjacent neurons.

11.1. *Linked Neurons* Sub-Window

This shows the list of connected neurons.

11.2. *Compartment Links* Sub-Window

This shows the list of compartments to which it is connected.

11.3. *Parallel Links* Sub-Window

This shows the list of parallel links to which the current neuron is connected.

11.4. *Parent Links* Sub-Window

This shows the list of parent links to which the current neuron is connected.

11.5. *Make Synaptic Links* Button

It asks if synaptic links will be added.

12. Moving Objects to and from Other Objects

Parameter values and objects can be moved to and from other objects easily. All it needs is to hold down the `option`-key in the keyboard while pressing the mouse key over the selected item. An arrow icon (⬇) will show up with an outline of the selected item showing. That selected item can be "dragged", i.e., you can hold down the mouse and move it to the window of another object. "Drop in" the selected item into the appropriate window or an editable box will make the corresponding changes. If a parameter value is dragged and dropped into another parameter box, that value will be become the new value. If an object is dragged and dropped into another object link, then these objects will become connected.

13. Neuronal Simulation Language Description Macro

The neuronal simulation language can be used to describe the morphology (and parameters) of the neuron and neural network, as well as describe the flow of control of the run-time environment of the simulation. Special *macros* are groups of commands specified by the user to execute the described commands and procedures for the execution and construction of the neural simulation. The *macros* are specified in the text-editor window, and the corresponding *macros* commands appear as menu-items appended to the *Macros* Menu. Thus, the *macros* can be executed by the user easily with a single menu command.

13. Neuronal Simulation Language

The *macros* language can be written in the text-editor (the lower partition of the main window). It can be saved into a file for future references.

The language is very similar to the existing Pascal programming language with most of the Pascal syntax for creating variables, and for control such as "*for-loops*", "*repeat-loops*" and "*while-loops*". Data types of *integers, reals* and *booleans* are defined as in Pascal. Subroutines that help modularize groups of commands are also available in the language, and are implemented as *procedures* and *functions*. Such functions and procedures can be used as "libraries" for building neurons from its components, and for specifying similar structures by calling these procedures repetitively with one change in the parameter-list, so that similar neurons can be built with different parameters specified by a procedure call.

In addition to the standard Pascal facility, the *macros* are constructed like a procedure as shown in the following example:

```
macro make_neuron_macro_command "make_10_neuron_macro";
var
  cerebellar_neuron: object;
  i: integer;
begin
  for i:= 1 to 10 do
  begin
    cerebellar_neuron:= new(neuron);    {create 10 neurons}
    cerebellar_neuron.name:= "Cerebellar Neuron";
  end;
end;

macro make_brain_macro_command "make_rat_brain_macro";
var
  rat_brain: object;
begin
  rat_brain:= new(brain);      {create a brain}
  rat_brain.name:= "Rat Brain";
end;

macro make_networ_macro_command "make_network_macro";
var
  left_cerebellar_network: object;
begin
  left_cerebellar_network:= new(brain); {create a network}
  left_cerebellar_network.name:= "Left Cerebellar Network";
end;
```

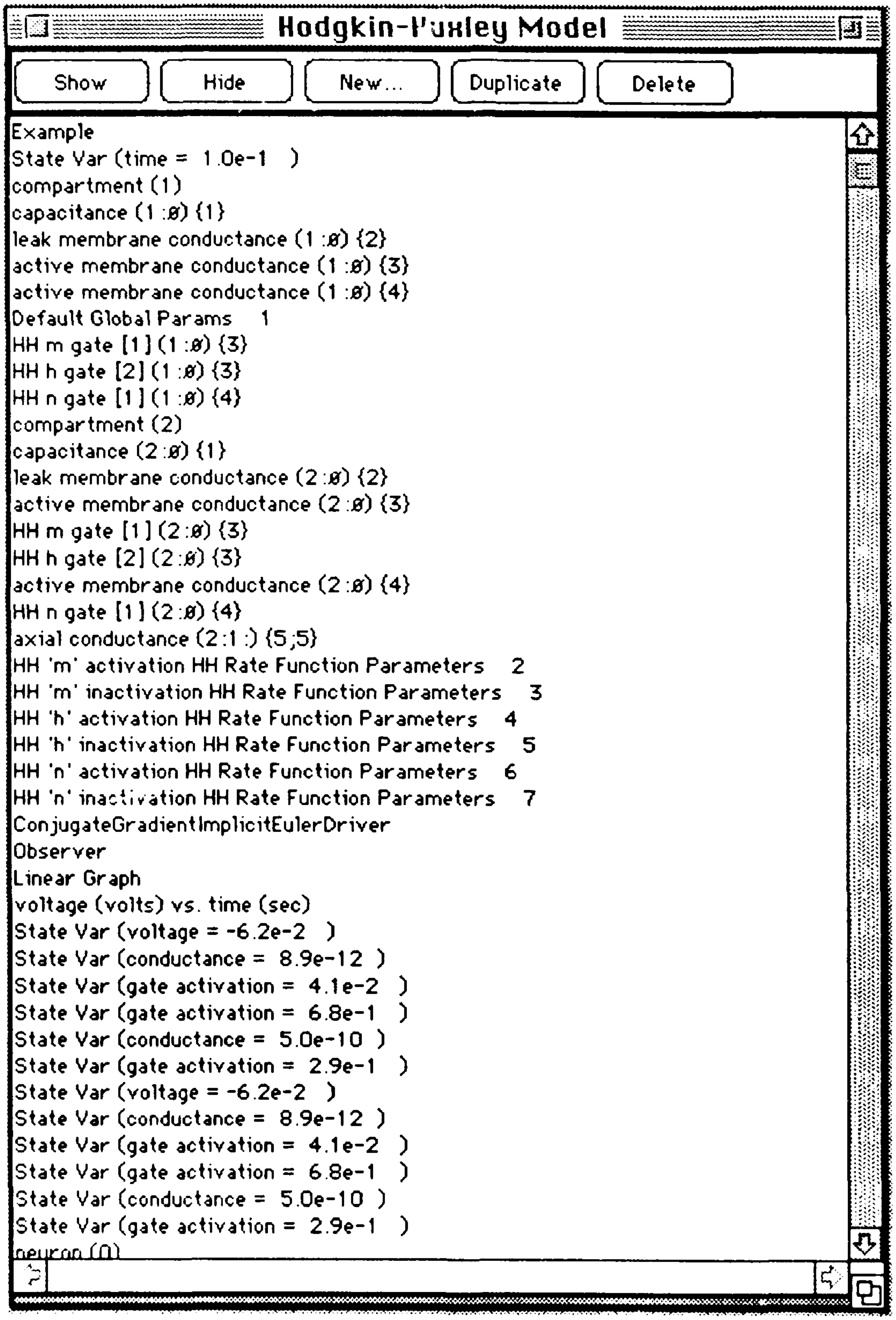
In the above example the first macro creates 10 cerebellar neurons, which will appear when the "*Check Syntax*" menu-item is executed. The *macro* is executed by pulling down the *Macro* Menu and selecting the "*make_10_neuron_macro*" menu-item.

When the "*make_rat_brain_macro*" is executed, the second macro creates a brain and calls it "Rat Brain".

When the "*make_network_macro*" is executed, the third macro creates a network and calls it "Left Cerebellar Network",

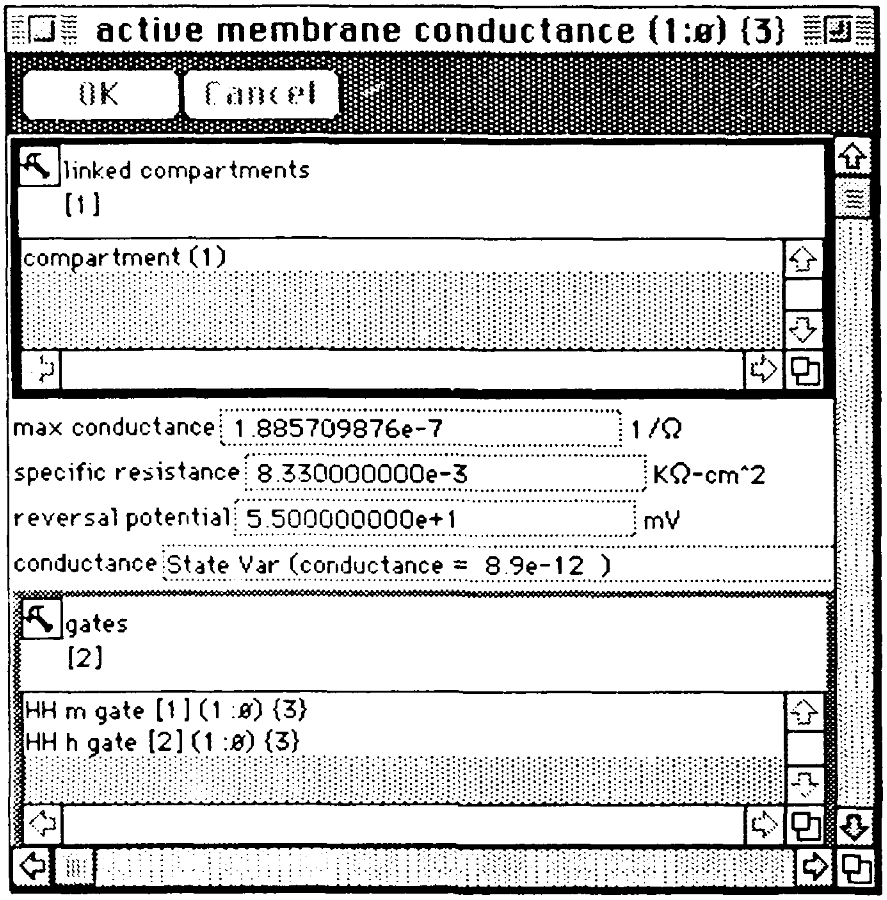
14. Example 2

The file*Hodgkin-Huxley Model* simulates a compartmental neuron with voltage-activated conductances. A list of all the objects in the simulation appears below.



```
Example
State Var (time =  1.0e-1  )
compartment (1)
capacitance (1 :0) {1}
leak membrane conductance (1 :0) {2}
active membrane conductance (1 :0) {3}
active membrane conductance (1 :0) {4}
Default Global Params   1
HH m gate [1] (1 :0) {3}
HH h gate [2] (1 :0) {3}
HH n gate [1] (1 :0) {4}
compartment (2)
capacitance (2 :0) {1}
leak membrane conductance (2 :0) {2}
active membrane conductance (2 :0) {3}
HH m gate [1] (2 :0) {3}
HH h gate [2] (2 :0) {3}
active membrane conductance (2 :0) {4}
HH n gate [1] (2 :0) {4}
axial conductance (2:1 :) {5;5}
HH 'm' activation HH Rate Function Parameters   2
HH 'm' inactivation HH Rate Function Parameters   3
HH 'h' activation HH Rate Function Parameters   4
HH 'h' inactivation HH Rate Function Parameters   5
HH 'n' activation HH Rate Function Parameters   6
HH 'n' inactivation HH Rate Function Parameters   7
ConjugateGradientImplicitEulerDriver
Observer
Linear Graph
voltage (volts) vs. time (sec)
State Var (voltage = -6.2e-2  )
State Var (conductance =  8.9e-12 )
State Var (gate activation =  4.1e-2  )
State Var (gate activation =  6.8e-1  )
State Var (conductance =  5.0e-10 )
State Var (gate activation =  2.9e-1  )
State Var (voltage = -6.2e-2  )
State Var (conductance =  8.9e-12 )
State Var (gate activation =  4.1e-2  )
State Var (gate activation =  6.8e-1  )
State Var (conductance =  5.0e-10 )
State Var (gate activation =  2.9e-1  )
neuron (0)
```

15. *Active Conductance* Window



active membrane conductance (1:ø) {3}

linked compartments
[1]

compartment (1)

max conductance 1.885709876e-7 1/Ω

specific resistance 8.330000000e-3 KΩ-cm^2

reversal potential 5.500000000e+1 mV

conductance State Var (conductance = 8.9e-12)

gates
[2]

HH m gate [1] (1 :ø) {3}
HH h gate [2] (1 :ø) {3}

Active Conductance includes a several new features that allow the user to simulated voltage gated activity.

15.1. *Linked Compartments* Sub-Window

Like the passive conductance windows described earlier, active conductances contain a sub-window which lists the compartments that it joins. In this case, there is only one compartment, and therefore the conductance is to ground by default.

15.2. *Max Conductance and Specific Resistance* Sub-Windows

The *max conductance* sub-window displays the maximum value of the conductance. This value is not directly editable but is computed from the *specific membrane resistance*. and the surface area adjacent membrane.

15.3. *Reversal Potential* Sub-Window

The reversal potential gives the potential difference across the conductance for which no current will flow.

15.4. *Conductance State Var* Sub-Window

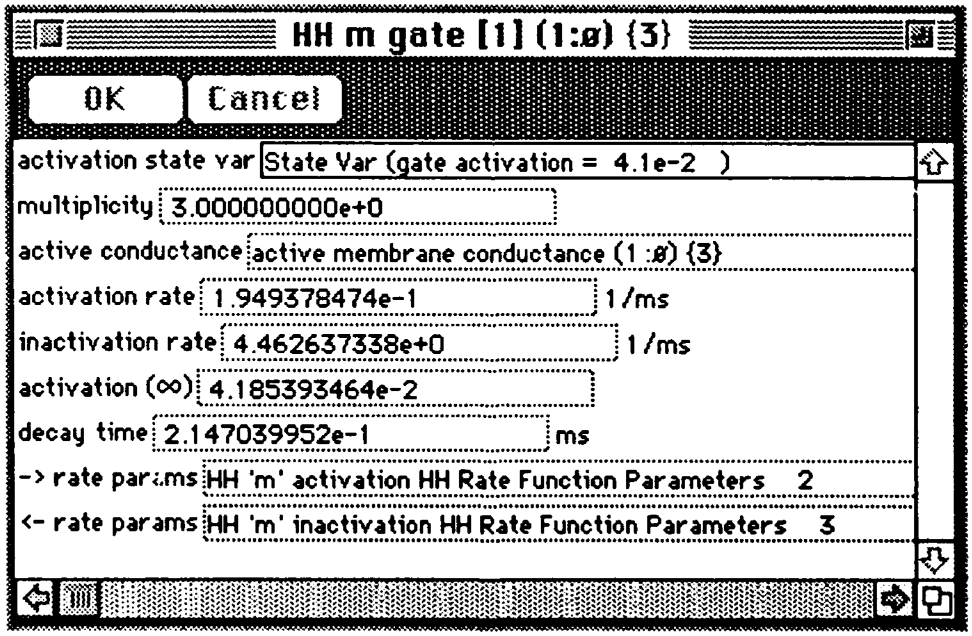
The *conductance state var* is a dynamical variable which records the current value of the conductance.

15.5. *Gate List* Sub-Window

The *Gate List* implements the active properties of *conductance*. They are described below.

16. *HH Gate* Window

The *HH Gate Window* describes a voltage activated gate. They are currently implemented in three varieties, *m, h,* and *n,* following the convention of Hodgkin and Huxley.



HH m gate [1] (1:ø) {3}

activation state var State Var (gate activation = 4.1e-2)
multiplicity 3.000000000e+0
active conductance active membrane conductance (1 :ø) {3}
activation rate 1.949378474e-1 1/ms
inactivation rate 4.462637338e+0 1/ms
activation (∞) 4.185393464e-2
decay time 2.147039952e-1 ms
-> rate params HH 'm' activation HH Rate Function Parameters 2
<- rate params HH 'm' inactivation HH Rate Function Parameters 3

16.1. *Activation State Var* Sub-Window

The *activation state var* is a dynamical variable which records the current value of the gate activation.

16.2.*Multiplicity* Sub-Window

The *multiplicity* sub-window specifies the power or weight of the gate in determining the total conductance through the channel. It is typically an integer and may be thought of as the number of identical (and independent) voltage activated gates in series.

16.3.*Active Conductance* Sub-Window

Reference to the *active conductance* object.

16.4.*Activation Rates* Sub-Windows

Gate activation is governed by an equation of the form:

$$dm/dt = \alpha(1-m) - \beta m$$

where the activation (inactivation) rate is given by α (β). These are also displayed in terms of the steady-state gate activation and the inverse instantaneous time constant.

16.5.*Activation Rate Function Parameters* Sub-Windows

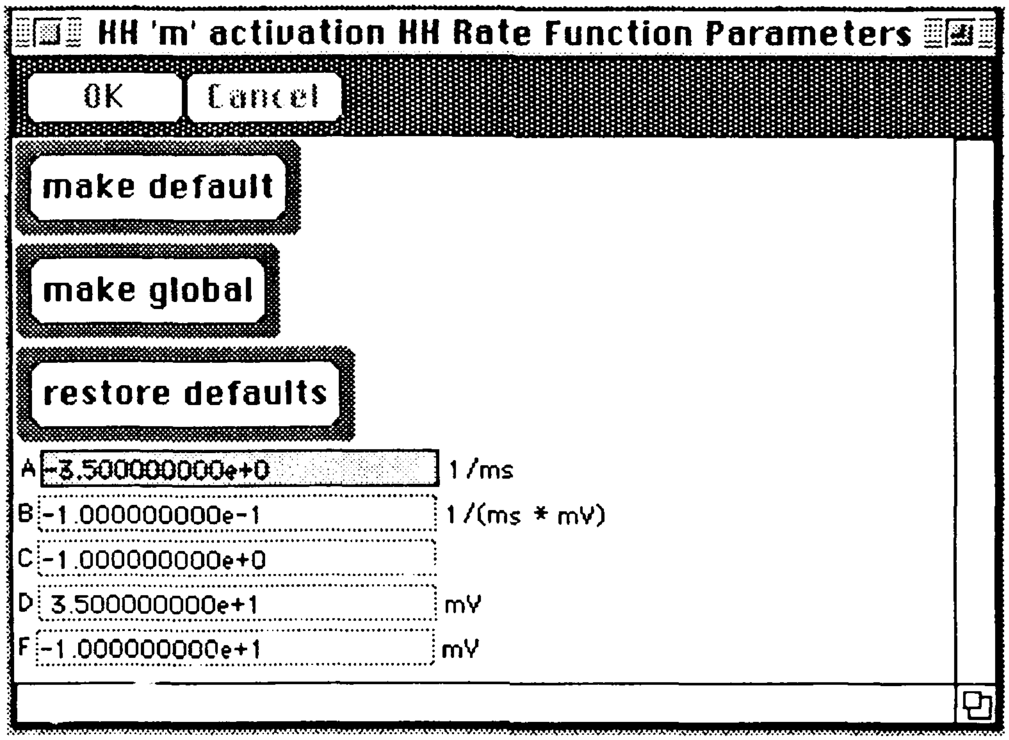
The instantaneous activation rate, α, is governed by an equation of the form:

$$\alpha(V) = (A + BV)/(C + exp[(V+D)/F]$$

where the parameters *A-F* are stored in separate parameters objects.

17. *HH Rate Function Parameters* Window

Gates in the model can have their own local parameters which can be shared by other gates by clicking on the *"make default"* button. This automatically causes all gates of the right type to use these parameters by default.



HH 'm' activation HH Rate Function Parameters

A -3.500000000e+0 1 /ms

B -1.000000000e-1 1 /(ms * mV)

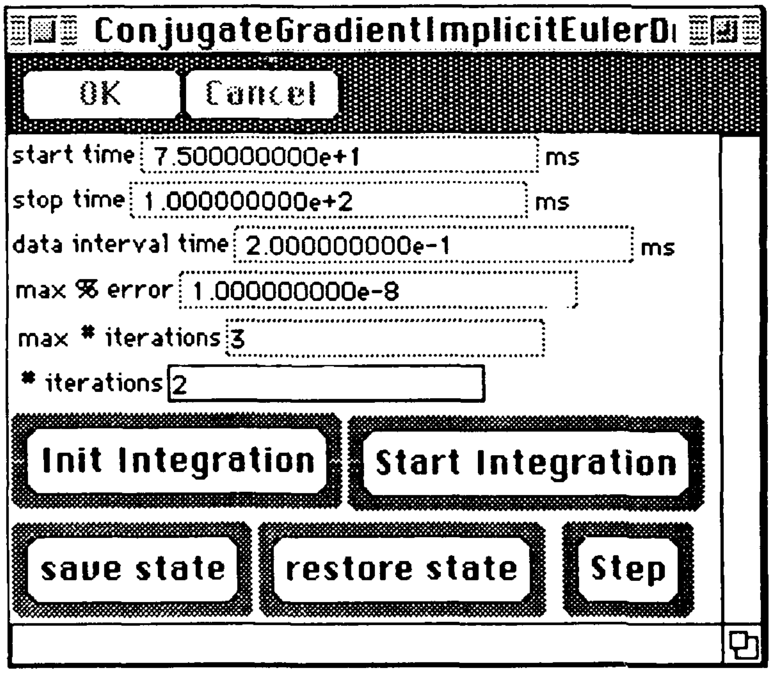
C -1.000000000e+0

D 3.500000000e+1 mV

F -1.000000000e+1 mV

18. *Integration Driver* Window

The *integration driver* advances the simulation through time.



ConjugateGradientImplicitEulerDı

start time 7.500000000e+1 ms

stop time 1.000000000e+2 ms

data interval time 2.000000000e-1 ms

max % error 1.000000000e-8

max # iterations 3

iterations 2

18.1 *stop, stop, and interval time* Sub-Windows

The *start, stop,* and *interval time* sub-windows control the duration and time resolution of the simulation.

18.2 *max % error, max # iterations, and # iterations* Sub-Windows

The *max % error* sub-window controls the number of conjugate gradient iterations per integration time step. Such iterations are performed until either the largest % change from the previous iteration is less than *max % error* or *max number iterations* has been exceeded. The *# iterations* window gives the actual number of iterations performed for that time step.

18.3 *Init and Start Integration* Command-Buttons

The *Init Integration* button initialized the integration. The integration must be initialize before the start button is clicked or an error message results. The *Start Integration* button begins the integration, which proceeds until the exit conditions are satisfied.

18.4 *Save and Restore State* Command-Buttons

The *Save* and *Restore* buttons save and restore the current or saved values of all dynamical variables.

18.5 *Step* Command-Button

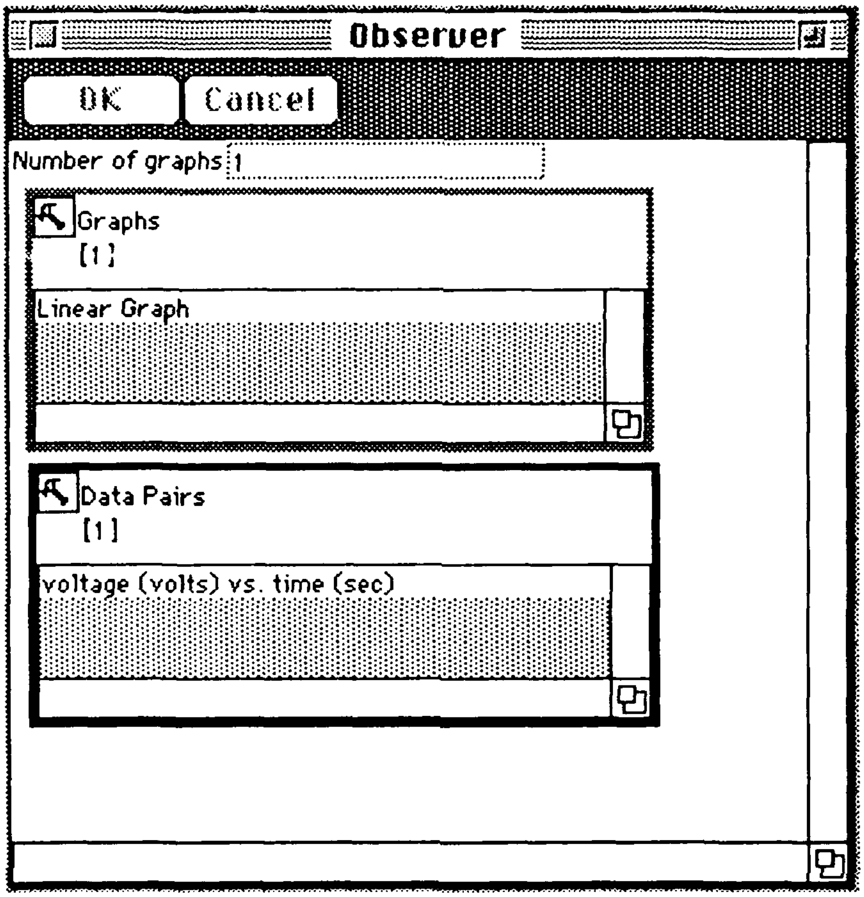
***Step* advances the simulation one time step.**

19. *Graphical Display*

The current version of the application has been enhanced with graphical capabilities as the following example shows.

19.1 *Observer* Window

Events in the application can be observed through an "Observer" window, which allows the user to organize his "Graphs" and "Data pairs". Currently, the observer can only be driven by updates to the global time, but this is expected to change in future implementations.



Number of graphs 1

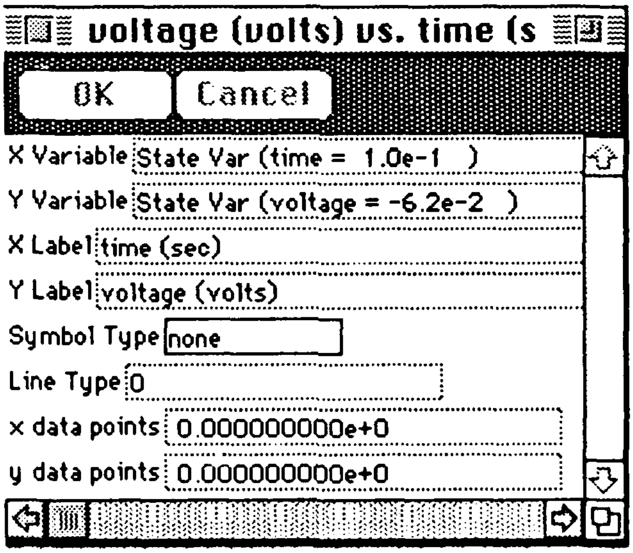
Graphs
[1]

Linear Graph

Data Pairs
[1]

voltage (volts) vs. time (sec)

19.2 *Data Pairs* and *State Vars*



X Variable State Var (time = 1.0e-1)

Y Variable State Var (voltage = -6.2e-2)

X Label time (sec)

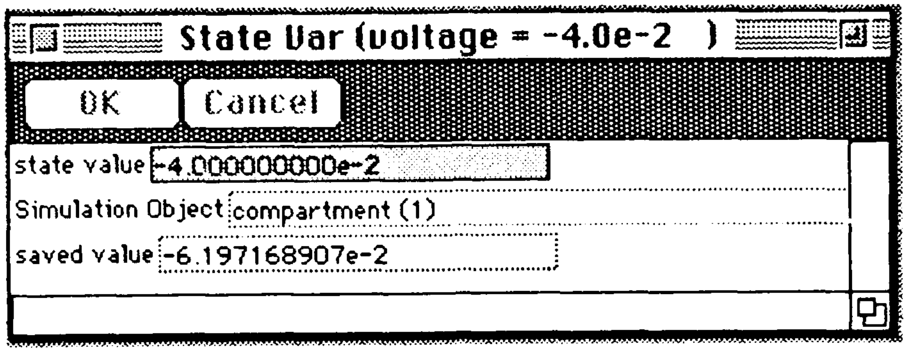
Y Label voltage (volts)

Symbol Type none

Line Type 0

x data points 0.000000000e+0

y data points 0.000000000e+0



state value -4.000000000e-2

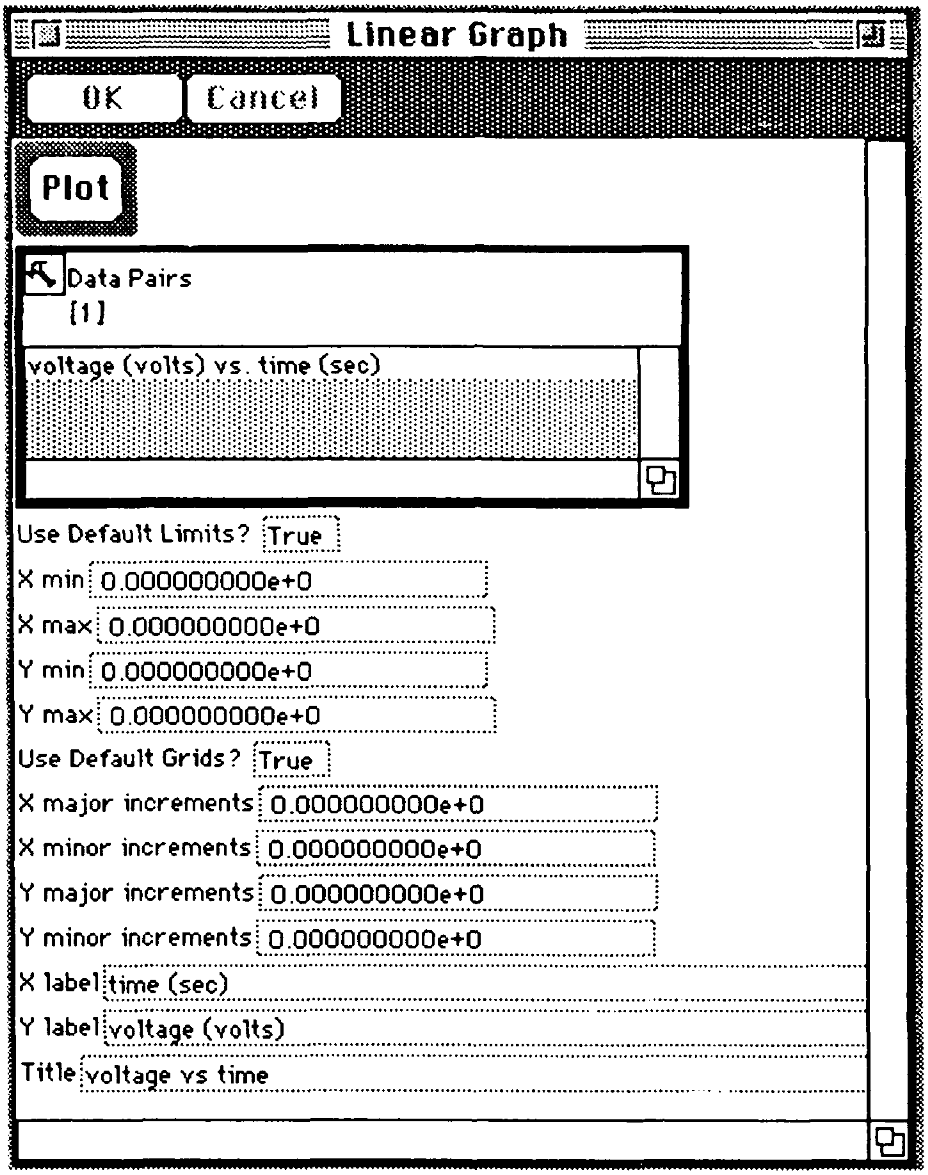
Simulation Object compartment (1)

saved value -6.197168907e-2

Dynamical variables (*State Vars*) are added to *Data Pairs* to construct coordinate pairs that can then be graphed. As show above, dynamical variables may also be edited directly by the user. Here, the membrane voltage has been forcibly depolarized by approximately 20.0 mV.

19.3 *Linear Graph* Window

Axis labels and title can be specified by the user by editing in the appropriate windows. Linear Graph uses default values for the other plot parameters unless overridden by the user.

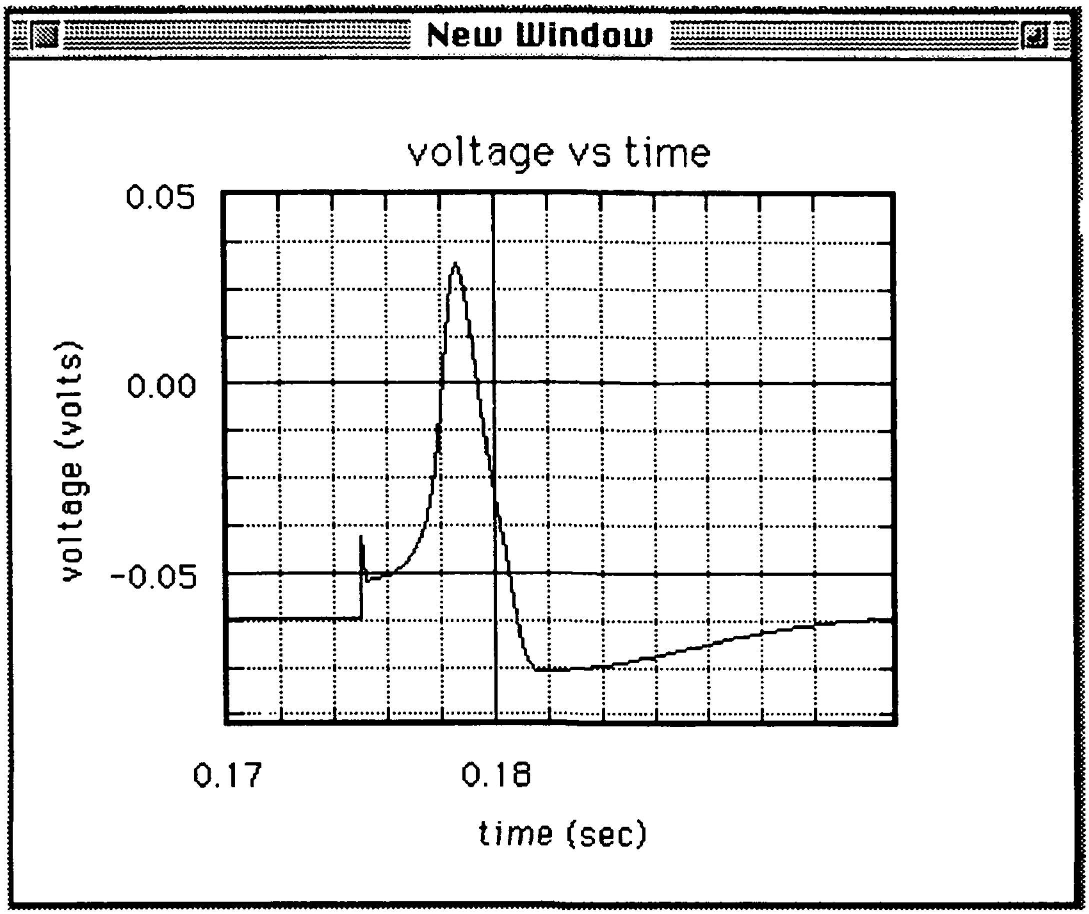


Data Pairs
[1]
voltage (volts) vs. time (sec)

Use Default Limits? True
X min 0.000000000e+0
X max 0.000000000e+0
Y min 0.000000000e+0
Y max 0.000000000e+0
Use Default Grids? True
X major increments 0.000000000e+0
X minor increments 0.000000000e+0
Y major increments 0.000000000e+0
Y minor increments 0.000000000e+0
X label time (sec)
Y label voltage (volts)
Title voltage vs time

19.4 *Plot* Window

Clicking *Plot* in the *Linear Graph* produces a plot of the *data pair* record. This forced depolarization is evident from the plot.


New Window
voltage vs time
voltage (volts)
0.05
0.00
-0.05
0.17
0.18
time (sec)